Vouch Protocol: Continuous State Verifiability for Autonomous AI Agents

W3C Community Group Report

Community Group Report, 31 May 2026

Specification revision: v1.6.2

This version: https://vouch-protocol.com/specs/CG-REPORT/2026-05-31/

Latest published version: https://vouch-protocol.com/specs/CG-REPORT/

Editor: Ramprasad Gaddam (Vouch Protocol), ram@vouch-protocol.com

Authors: Ramprasad Gaddam (Vouch Protocol), Manu Sporny (Digital Bazaar)

Co-sponsor: Manu Sporny (Digital Bazaar), msporny@digitalbazaar.com

Participate:

- W3C Credentials Community Group mailing list: public-credentials@w3.org (archives: https://lists.w3.org/Archives/Public/public-credentials/)

- GitHub issues: https://github.com/vouch-protocol/vouch/issues

- Discord: https://discord.gg/mMqx5cG9Y

About the Editor

Ramprasad Anandam Gaddam is Director of AI and Machine Learning Engineering at Optum, UnitedHealth Group with extensive experience in designing and operating AI systems at scale within highly regulated industries. Ramprasad's professional focus has centered on the intersection of autonomous AI systems and the regulatory, liability, and accountability frameworks that govern banking, healthcare, and other safety-critical sectors. This perspective, that of a practitioner who must answer regulators, auditors, and patients/customers when an AI system acts, has shaped the design priorities of this specification.

Ramprasad's interest in agent identity standards stems from a concrete operational problem: as autonomous AI agents are deployed to make real-world decisions on behalf of organizations and individuals, existing authentication frameworks (API keys, OAuth

bearer tokens, session cookies) do not provide the cryptographic chain-of-custody, intent attestation, or behavioral attestation that regulated environments require. Vouch Protocol began as an attempt to fill this gap using primitives that are already familiar to the W3C community: Decentralized Identifiers, Verifiable Credentials, and Data Integrity proofs.

Ramprasad has 20 years of experience in Healthcare & Manufacturing industries, with over 16 years in healthcare. He is a Master Inventor with 20 patents filed in Blockchain, Cybersecurity, Artificial Intelligence, Healthcare processes & Information Technology. He has 60 defensive disclosures with Vouch Protocol to encourage open adoption. He is a member of other standard boards like The Linux Foundation, Coalition for Content Provenance & Authenticity (C2PA), Content Authenticity Initiative (CAI), Decentralized Identity Foundation (DIF) and IEEE. He is currently employed with Optum, UnitedHealth Group and heads Responsible AI Acceleration Service for Optum Health. He was inducted into UHG Inventor Hall of Fame twice, is a member of Patent Review Boards, Optum National Council & AI Sub-council. He is also UHG AI Review Board member. He is winner of Optum Technology Make IT Happen award twice; it is an annual recognition award from Optum Technology that honors team members who help move tech forward by advancing tech pillars.

Abstract

This specification defines the **Vouch Protocol**, an open standard for establishing **continuous state verifiability** of autonomous AI agents, a layer that sits beneath, and complements, agent identity and delegation specifications.

Where existing agent identity specifications answer *“who is this agent and what are they authorized to do?”*, Vouch Protocol answers the operational follow-on questions: *“is this agent’s runtime state still aligned with its authorization?”*, *“how do we maintain trust as the agent operates over time?”*, and *“what does post-quantum migration look like for an agent identity layer?”*

The protocol uses W3C Verifiable Credentials secured with W3C Data Integrity proofs (`eddsa-jcs-2022` cryptosuite), W3C Decentralized Identifiers as the identity format, and the Multikey verification method to support cryptographic agility. It introduces several novel architectural mechanisms: the Identity Sidecar pattern for LLM key isolation, recursive delegation chains with explicit resource binding, continuous trust maintenance via the Heartbeat Protocol, federated validator quorum, and an optional dual-proof post-quantum profile in which a credential carries two independent Data Integrity proofs (one `eddsa-jcs-2022`, one `mldsa44-jcs-2026`) on the same JCS-canonicalized payload bytes.

Status of This Document

This document is submitted to the W3C **Credentials Community Group** (`public-credentials@w3.org`) for incubation as a Community Group Report. It is not a W3C Standard nor is it on the W3C Standards Track. Publication as a Community Group Report does not imply endorsement by the W3C Membership. Contributions to this document are governed by the W3C Community Contributor License Agreement (CLA), under which a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

Transition pathways under consideration include the W3C Verifiable Credentials Working Group and the Data Integrity Working Group, contingent on demonstrated implementer interest and Credentials Community Group judgment.

Table of Contents

1. Introduction
2. Terminology
3. Conformance
4. Data Model
5. Vouch Credential Format
6. Identity Model
7. Signing Operations
8. Verification Operations
9. Delegation Chains
10. Identity Sidecar Pattern
11. Credential Lifecycle and the Heartbeat Protocol
12. Security Considerations
13. Crypto-Agility and Quantum-Safe Profile
14. Privacy Considerations
15. State Verifiability (Informative)
16. Implementer Interest and Charter Sponsors
17. Conformance Levels
18. Acknowledgements
19. Appendix A: Relationship to Existing Standards
20. Appendix B: IANA Considerations
21. Appendix C: Test Vectors
22. References

---

1. Introduction

1.1 Problem Statement

Autonomous AI agents are increasingly deployed to take real-world actions across regulated sectors, executing financial transactions, accessing protected health information, authorizing insurance claims, submitting regulatory filings, and operating within critical infrastructure. The operational and regulatory environment for these deployments includes (non-exhaustively):

- **Banking and capital markets**: U.S. SR 11-7 model risk management, FFIEC guidance on AI, SEC Rule 15c3-5, MiFID II algorithmic trading rules, and FINRA notices on AI accountability.
- **Healthcare and life sciences**: HIPAA, HITECH, FDA Software-as-a-Medical-Device guidance, EU MDR, 21 CFR Part 11 for electronic records, and ICH E6(R3) for clinical trial integrity.
- **Insurance**: NAIC Model Bulletin on AI (2024) and NYDFS Cybersecurity Regulation (Part 500).
- **Government and defense**: FedRAMP, CMMC 2.0, U.S. Executive Order 14110, NIST CSF 2.0.
- **Critical infrastructure**: NERC CIP for the bulk power system, TSA pipeline directives.
- **Cross-cutting (EU)**: The EU AI Act, applicable from 2025, imposes auditability and human-oversight obligations on high-risk AI systems.
- **Cross-cutting (post-quantum migration)**: NIST CNSA 2.0, U.S. National Security Memorandum 10, and CNSSP-15 require migration to quantum-resistant cryptography on defined timelines.

Existing authentication mechanisms (API keys, Bearer tokens, OAuth 2.0) were designed for human users and stateless services. Newer agent identity specifications address *who* an agent is and *what* it has been authorized to do. Operators in the regulated sectors above have additional, currently-unmet requirements:

1. **Identity Binding**: Cryptographic proof that a specific agent, controlled by a specific principal, performed a specific action.
2. **Intent Attestation**: Non-repudiable attestation of the action the agent authorized, bound to the resource the action targets, not just the action's name.
3. **Delegation Accountability**: A verifiable chain from human principal to the agent (and any sub-agents) that actually executed the action, with explicit per-link resource scope.
4. **Continuous State Verifiability**: Mechanisms to continuously verify that an agent's runtime behavior remains aligned with its authorized intent, beyond a single point-in-time authentication.

5. **Post-Quantum Migration Path**: A concrete roadmap from current elliptic-curve signatures to quantum-resistant signatures, addressing harvest-now-decrypt-later and retroactive forgery threats.

1.2 Design Goals

The Vouch Protocol is designed with the following goals:

- **Standards-Aligned**: Built on W3C Verifiable Credentials, W3C Data Integrity proofs (`eddsa-jcs-2022`), W3C Decentralized Identifiers, and Multikey verification methods. No new cryptographic primitives are introduced where existing standards suffice.
- **Crypto-Agile**: The signature algorithm is selected per-deployment via Multikey and the cryptosuite mechanism. Migration to post-quantum signatures (ML-DSA-44) is supported as an optional dual-proof profile in which a credential carries two independent Data Integrity proofs (`eddsa-jcs-2022` and `mldsa44-jcs-2026`) on the same JCS-canonicalized payload bytes, with reference implementations in Python, TypeScript, and Go and published cross-implementation test vectors.
- **Framework-Agnostic**: Operates with any AI agent framework, including the Model Context Protocol (MCP), LangChain, CrewAI, AutoGPT, AutoGen, and custom frameworks, by binding identity to credentials rather than to a specific transport.
- **Fail-Secure**: The default state is untrusted. Identity must be cryptographically proven; there is no fallback to bearer-token or session-cookie authentication.
- **LLM-Safe**: Private keys are never exposed to the LLM context window. The Identity Sidecar pattern (Section 10) is normative.
- **Human-Auditable**: Vouch credentials are readable JSON. Cryptographic proofs attach as sibling objects rather than encasing the payload in opaque encoded strings, enabling direct human inspection during audit and incident response.

1.3 Scope

This specification covers: - The Vouch credential format and its required claims. - Agent identity using W3C Decentralized Identifiers, with `did:web` and `did:key` as the primary supported methods. - Signing and verification operations using the `eddsa-jcs-2022` Data Integrity cryptosuite. - Intent-bound delegation chains for multi-agent systems. - The Identity Sidecar architectural pattern. - Credential lifecycle management via the Heartbeat Protocol. - Credential and key revocation, including credential-level status via W3C BitstringStatusList (shipped across Python, TypeScript, and Go SDKs with a cross-language test vector) and a complementary DID-level revocation registry (`vouch.revocation`). - An optional dual-proof post-quantum profile in which a credential carries two independent Data Integrity proofs (one `eddsa-jcs-2022`, one `mldsa44-jcs-2026`) on the same JCS-canonicalized payload bytes, shipped with cross-language reference implementations and test vectors. - An informative description of the State Verifiability layer.

This specification does NOT cover: - A normatively-defined reputation scoring algorithm. The Python SDK includes an informative reference algorithm (exponential decay toward baseline, plus clamped action deltas) for illustration; implementers MAY use it, supply their own, or omit reputation entirely. - Confidentiality of agent intent payloads (a future companion specification will define an optional confidentiality profile for sensitive intents). - Media provenance (see companion C2PA integration specification). - Specific AI framework integration APIs (see SDK documentation).

---

2. Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [BCP 14] when, and only when, they appear in all capitals, as shown here.

Agent An autonomous software entity (typically an AI/LLM-based system) that performs actions on behalf of a principal.

Principal The human user or organization that owns and is accountable for an agent’s actions.

Vouch Credential A W3C Verifiable Credential conforming to this specification, secured with a Data Integrity proof. It carries the agent’s identity, intent payload, optional delegation chain, and optional reputation indicators.

Passport The decoded and verified Vouch Credential. Represents the agent’s verified identity and authorized intent at a point in time.

DID (Decentralized Identifier) A globally unique identifier as defined by [W3C DID Core], used to identify agents and principals.

Multikey A verification method type as defined in the [W3C Controlled Identifiers] specification, encoding public key material as a multibase + multicodec string. Algorithm-agnostic across Ed25519, ECDSA P-256, secp256k1, ML-DSA-44, and others.

Identity Sidecar A deterministic co-process that holds the agent’s private keys and performs signing operations, isolated from the non-deterministic LLM context.

Delegation Chain An ordered sequence of signed authorization steps from a root principal to the executing agent, establishing a cryptographic chain of custody. Each link binds the authority to a specific resource.

Heartbeat A periodic credential renewal request that hash-links to its predecessor and includes behavioral attestation, forming a continuous trust chain.

Validator A service that evaluates agent behavior and issues renewed Session Vouchers as part of the Heartbeat Protocol.

Session Voucher A short-lived credential issued by a Validator quorum, carrying trust decay parameters and scope restrictions.

Trust Entropy The continuous exponential decay of credential trust level over time, enabling risk-proportional access decisions.

State Verifiability The property that an agent's runtime state, including authorization scope, behavioral history, and continuity, can be cryptographically verified at any point in time.

---

3. Conformance

A conforming **Vouch Signer** implementation: - MUST issue Vouch Credentials as W3C Verifiable Credentials with a Data Integrity proof using the `eddsa-jcs-2022` cryptosuite [VC-DI-EDDSA]. - MUST express verification methods using Multikey [W3C Controlled Identifiers]. - MUST include the REQUIRED claims defined in Section 5. - MUST set `proof.type` to `"DataIntegrityProof"` and `proof.cryptosuite` to `"eddsa-jcs-2022"`. - MUST canonicalize the credential payload using JSON Canonicalization Scheme (JCS) [RFC 8785] prior to signing. - MAY additionally emit a second Data Integrity proof using the post-quantum cryptosuite defined in Section 13 (the dual-proof profile).

A conforming **Vouch Verifier** implementation: - MUST verify Data Integrity proofs according to [VC-DI-EDDSA]. - MUST validate temporal claims (`validFrom`, `validUntil`) with configurable clock skew tolerance. - MUST reject credentials where `validUntil` is in the past or `validFrom` is in the future (subject to clock skew tolerance). - MUST resolve the agent's verification method via DID resolution as specified in Section 8.3. - SHOULD support DID resolution for `did:web` identifiers. - MAY support additional DID methods. - MAY additionally verify the second `mldsa44-jcs-2026` proof on credentials that use the dual-proof post-quantum profile defined in Section 13.

A conforming **Vouch Agent** implementation: - MUST use the Identity Sidecar pattern (Section 10) to isolate private keys from the LLM context window. - MUST include an intent payload in every Vouch Credential. - SHOULD support delegation chains when operating as a sub-agent.

---

4. Data Model

4.1 Agent Identity

An agent identity consists of:

Component	Description	Format
Private Key	Signing key (Ed25519 by default; see Section 13 for the dual-proof post-quantum profile)	Raw bytes; encoded only at rest
Public Key	Verification key	Multibase-encoded, with multicodec prefix indicating the algorithm
DID	Decentralized Identifier	`did:web:<domain>` or `did:key:<multibase>`

4.2 Key Storage

Private keys MUST be stored encrypted at rest. The RECOMMENDED encryption scheme is:

- **Key Derivation**: Scrypt (n=16384, r=8, p=1) with 16-byte random salt
- **Encryption**: ChaCha20-Poly1305 (AEAD) with 12-byte random nonce
- **File Permissions**: 0600 (owner read/write only)

Keys MAY be backed by a hardware security module (HSM), TPM, or platform key store.

4.3 DID Document

Agents using `did:web` MUST host a DID Document at `https://<domain>/.well-known/did.json`:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/multikey/v1"
  ],
  "id": "did:web:agent.example.com",
  "verificationMethod": [{
    "id": "did:web:agent.example.com#key-1",
    "type": "Multikey",
    "controller": "did:web:agent.example.com",
    "publicKeyMultibase": "z6MkrJVnaZkeFzdQyMZu1cgjg7k1pZZ6pvBQ7XJPt4swbTQ2"
  }],
  "authentication": ["did:web:agent.example.com#key-1"],
  "assertionMethod": ["did:web:agent.example.com#key-1"]
}
```

The Multikey format is algorithm-agnostic. The same `verificationMethod` slot supports Ed25519 (multicodec `ed25519-pub`, varint `0xed01`), ECDSA P-256 (`p256-pub`),

ML-DSA-44 (`mldsa44-pub`, see Section 13), and additional algorithms as registered. A DID Document MAY contain multiple verification methods to support the dual-proof post-quantum profile (Section 13), with one verification method per cryptosuite the issuer is prepared to sign under.

---

5. Vouch Credential Format

5.1 Overview

A Vouch Credential is a W3C Verifiable Credential [VC-DATA-MODEL-2.0] secured by a Data Integrity proof [VC-DATA-INTEGRITY] using the `eddsa-jcs-2022` cryptosuite [VC-DI-EDDSA].

5.2 Example

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://vouch-protocol.com/contexts/v1"
  ],
  "id": "urn:uuid:550e8400-e29b-41d4-a716-446655440000",
  "type": ["VerifiableCredential", "VouchCredential"],
  "issuer": "did:web:agent.example.com",
  "validFrom": "2026-04-26T10:00:00Z",
  "validUntil": "2026-04-26T10:05:00Z",
  "credentialSubject": {
    "id": "did:web:agent.example.com",
    "vouchVersion": "1.0",
    "intent": {
      "action": "read_database",
      "target": "users_table",
      "resource": "https://api.example.com/v1/users"
    }
  },
  "proof": {
    "type": "DataIntegrityProof",
    "cryptosuite": "eddsa-jcs-2022",
    "created": "2026-04-26T10:00:00Z",
    "verificationMethod": "did:web:agent.example.com#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "z3FXQjecWufY46..."
  }
}
```

Two notes on this example:

- **The `id` property is OPTIONAL.** Per VC 2.0, a Verifiable Credential is not required to carry an `id`. Vouch Credentials whose verifier-side audit chain is the

`proof.created` + `issuer` + JCS-canonical bytes (the common case for ephemeral, short-lived intent assertions) MAY omit `id`. A `urn:uuid` value is RECOMMENDED for credentials that are persisted, replayed, or correlated across audit logs.

- **The `intent` object is an authorization capability in shape.** A reader familiar with ZCAP-LD will recognize the `action` / `target` / `resource` triple as an unscoped invocation of a capability. Vouch deliberately uses RFC 8785 JCS canonicalization rather than JSON-LD canonicalization, and binds each delegation link to an explicit `resource` URI (Section 9); Appendix A summarizes the trade-offs and the prior-art relationship to ZCAP-LD.

5.3 Required Properties

Property	Required	Type	Description
`@context`	REQUIRED	Array	MUST contain `https://www.w3.org/ns/credentials/v2` and the Vouch context URL.
`id`	OPTIONAL	URI	A unique identifier for the credential. RECOMMENDED (UUID URN form) for credentials that are persisted, replayed, or correlated across audit logs; MAY be omitted for ephemeral, short-lived intent credentials. Aligns with VC 2.0, which makes `id` optional.
`type`	REQUIRED	Array	MUST contain `"VerifiableCredential"` and `"VouchCredential"`.
`issuer`	REQUIRED	DID	The DID of the issuing agent.
`validFrom`	REQUIRED	XML Schema datetime	Timestamp before which the credential is not valid.

Property	Required	Type	Description
`validUntil`	REQUIRED	XML Schema datetime	Expiration. RECOMMENDED maximum: 300 seconds after `validFrom`.
`credentialSubject`	REQUIRED	Object	Contains agent intent. See Section 5.4.
`proof`	REQUIRED	Object	Data Integrity proof. See Section 5.5.
`credentialStatus`	OPTIONAL	Object	Revocation reference (e.g., StatusList2021 / BitstringStatusList).

5.4 Credential Subject

The `credentialSubject` object MUST contain:

Property	Required	Type	Description
`id`	REQUIRED	DID	The DID of the agent the credential describes. For self-issued credentials, equal to `issuer`.
`vouchVersion`	REQUIRED	String	Protocol version. Current: `"1.0"`.
`intent`	REQUIRED	Object	The action being authorized. See Section 5.4.1.
`reputationScore`	OPTIONAL	Integer (0-100)	Agent's self-reported reputation score.
`delegationChain`	OPTIONAL	Array	Ordered list of delegation links (see Section 9).

5.4.1 Intent Payload

The `intent` object describes the agent's authorized action. It MUST include:

Field	Required	Description
`action`	REQUIRED	The action being performed (e.g., `"read_database"`, `"execute_trade"`, `"submit_claim"`)
`target`	REQUIRED	The semantic target of the action (e.g., `"users_table"`, `"order:42"`)
`resource`	REQUIRED	A URI identifying the specific resource (e.g., `"https://api.example.com/v1/users"`, a database connection string, an MCP tool URI)

The `resource` field is REQUIRED to support precise authorization-capability semantics (see Section 9 and [PAD-021] Inverse Capability Protocol). Implementations MUST NOT issue Vouch Credentials with vague or missing resource bindings.

Application-specific fields MAY be included.

5.5 Data Integrity Proof

The `proof` object MUST conform to [VC-DATA-INTEGRITY] using the `eddsa-jcs-2022` cryptosuite [VC-DI-EDDSA]:

Field	Required	Description
`type`	REQUIRED	MUST be `"DataIntegrityProof"`
`cryptosuite`	REQUIRED	MUST be `"eddsa-jcs-2022"` for the classical proof. For the dual-proof post-quantum profile (Section 13), a second proof object in the same `proof` array uses `"mldsa44-jcs-2026"`. The transitional composite identifier `"hybrid-eddsa-mldsa44-jcs-2026"` is retained for v1.6.x interop only.
`created`	REQUIRED	XML Schema datetime

Field	Required	Description
`verificationMethod`	REQUIRED	DID URL of the verification method (e.g., `did:web:agent.example.com#key-1`)
`proofPurpose`	REQUIRED	MUST be `"assertionMethod"`
`proofValue`	REQUIRED	Multibase-encoded signature

The dual-proof post-quantum profile, in which Ed25519 and ML-DSA-44 proofs are computed over the same RFC 8785 JCS-canonicalized bytes, is described in [PAD-040] (Dual-Proof Same-Canonical-Bytes Property). Algorithm-agnostic verification method resolution via Multikey multicodec discrimination, which lets a verifier route a credential to the correct cryptographic backend without negotiation, is described in [PAD-041] (Algorithm-Agnostic Verification via Multikey Multicodec).

5.6 Transport

Vouch Credentials are transmitted in the HTTP body of the request to which they pertain, with the credential bound by hash to the request body. Transport-binding details:

```
POST /api/resource HTTP/1.1
Host: api.example.com
Content-Type: application/vc+vouch

{
  "@context": [...],
  "type": ["VerifiableCredential", "VouchCredential"],
  ...
  "credentialSubject": {
    "id": "did:web:agent.example.com",
    "vouchVersion": "1.0",
    "intent": { "action": "read_database", "target": "users_table",
"resource": "https://api.example.com/v1/users" }
  },
  "proof": { ... }
}
```

For deployments where the API request body must remain unmodified, the credential MAY be transmitted via the `Vouch-Credential` HTTP header (Base64URL-encoded). Implementations SHOULD prefer the body-bound form to avoid header size constraints, particularly when using the dual-proof post-quantum profile (Section 13).

5.6.1 Transport Independence and Asynchronous Patterns

A Vouch Credential is a JSON payload. The specification defines its **format** and **canonicalization**, not its **delivery channel**. While the synchronous HTTP request/response pattern shown above is the most common transport, the Vouch Credential format is transport-independent and works unchanged across the following operational patterns:

- **Synchronous request/response (the default).** The credential travels in the HTTP request body or the `Vouch-Credential` header. The work it authorizes completes within the lifetime of a single HTTP exchange. This is the common case for fast operations (read a record, submit a small claim, place a trade) and is fully covered by the format defined above.

- **Asynchronous submit-and-poll.** The agent submits a credential authorizing the *initiation of a long-running job* (`intent.action = "start_report_job"`). The server returns a job identifier. When the agent later polls for the result, a separate Vouch Credential authorizes that polling action (`intent.action = "read_job_result"`, `intent.target = job_id`). Two credentials, two intents, bound to two distinct resources; each verifiable independently.

- **Webhook / callback.** The worker server, on completion of the long-running job, issues its own outbound HTTP request (the webhook callback) carrying its own Vouch Credential. The callback's credential identifies the worker, the result, and any chain of delegation back to the original requester. The original credential authorized the submission; the callback credential authorizes the response.

- **Message queue.** When delivery is brokered through a message queue (Apache Kafka, AWS SQS, RabbitMQ, Google Pub/Sub), the Vouch Credential is part of the queue message payload. The queue transports it; the receiver verifies it on dequeue. The format is identical to the HTTP body form.

- **Filesystem / object-store artifact.** When the work product is a stored artifact (an uploaded report in S3, a generated image in C2PA-bound form, an audit log entry in a write-only ledger), the Vouch Credential is stored alongside the artifact or embedded in its manifest. Verification occurs at retrieval time. Recursive composition with C2PA Content Credentials is described in PAD-014 and PAD-028.

- **Cryptographically-anchored policy decisions (informative).** Where the credential records a deterministic policy decision rather than an intent to act, the bridge described in PAD-059 (Vouch-Amnesia Attestation Bridge) signs the decision asynchronously: a synchronous policy evaluator produces a block / attest / allow verdict at the moment of egress, and an asynchronous signer posts the resulting Verifiable Credential to a local or remote audit log without delaying the operation being evaluated. The credential format is unchanged; the asynchronous signing pattern is the operational extension.

The recursive composition pattern across these transports is the same: an agent that initiates work in pattern (1) and receives an asynchronous result in pattern (2) through (5) can chain the two credentials as a delegation chain (Section 9), so a verifier can prove “agent A authorized job J at time T_1, and worker W returned result R at time T_2, both with cryptographic continuity to the same principal.” No specification change is required to support this composition; it follows directly from the credential format and the delegation chain semantics.

Implementations SHOULD NOT assume synchronous HTTP semantics when designing a Vouch-aware system. The credential’s expiration window (`validUntil`) and its bound `intent.resource` are sufficient to express short-lived synchronous authorization; longer-lived asynchronous authorization is expressed by issuing additional credentials for each work step rather than by stretching a single credential’s validity. Compositions of multiple short-lived credentials are preferable to single long-lived credentials in all of the above patterns.

---

6. Identity Model

6.1 DID Methods

Vouch Protocol supports multiple DID methods. Implementations MUST support `did:web` and SHOULD support `did:key`.

6.1.1 did:web

The `did:web` method [DID Web] uses DNS as the trust anchor.

- **Format**: `did:web:<domain>`
- **Resolution**: HTTPS GET to `https://<domain>/.well-known/did.json`
- **Advantages**: Zero-cost, instant setup, compatible with existing IT infrastructure.
- **Limitations**: Centralized trust (depends on DNS/HTTPS).

6.1.2 did:key

The `did:key` method encodes the public key directly in the identifier.

- **Format**: `did:key:<multibase-encoded-public-key>`
- **Resolution**: The public key is derived directly from the identifier.
- **Advantages**: Self-contained; no external resolution required.
- **Limitations**: No key rotation without changing the DID.

6.2 Key Generation

Implementations MUST generate Ed25519 keys using a cryptographically secure random number generator (CSPRNG). The public key MUST be encoded as a `Multikey`

with the appropriate multicodec prefix (`0xed01` for Ed25519). Private keys MUST NOT be transmitted, logged, or exposed to the LLM context window.

---

7. Signing Operations

7.1 Credential Issuance

To issue a Vouch Credential, the signer:

1. Constructs the credential object with all REQUIRED fields (Section 5).
2. Constructs an unsigned `proof` object containing all proof properties EXCEPT `proofValue`.
3. Adds the unsigned proof to the credential.
4. Canonicalizes the entire credential (including the unsigned proof) using JSON Canonicalization Scheme (JCS) [RFC 8785].
5. Computes a SHA-256 digest of the canonicalized bytes.
6. Signs the digest using the Ed25519 private key.
7. Adds the multibase-encoded signature as `proof.proofValue`.

This procedure conforms to [VC-DI-EDDSA] §3.1 (`eddsa-jcs-2022` cryptosuite).

7.2 Algorithm Restriction

This specification mandates the `eddsa-jcs-2022` cryptosuite for v1.0 conformance. Implementations MAY additionally support the post-quantum `mldsa44-jcs-2026` cryptosuite as a second Data Integrity proof on the same credential (the dual-proof profile of Section 13).

Algorithm negotiation in the wire format is not supported. The cryptosuite is unambiguously declared by `proof.cryptosuite`.

---

8. Verification Operations

8.1 Credential Verification

To verify a Vouch Credential, the verifier:

1. Parses the credential JSON.
2. Extracts the `issuer` DID.
3. Resolves the `proof.verificationMethod` to obtain the public key (Section 8.3).
4. Removes `proof.proofValue` from the credential.
5. Canonicalizes the remaining credential (including the unsigned proof) using JCS [RFC 8785].

6. Computes the SHA-256 digest of the canonicalized bytes.
7. Verifies the Ed25519 signature against the public key.
8. Validates temporal claims:
    - `validUntil` > current time (with clock skew tolerance).
    - `validFrom` ≤ current time (with clock skew tolerance).
9. Validates `credentialStatus` if present (Section 11.2).
10. Returns the verified Passport.

8.2 Clock Skew

Implementations SHOULD allow a configurable clock skew tolerance. The RECOMMENDED default is 30 seconds. The maximum permitted skew is 120 seconds.

8.3 DID Resolution

For verification, the verifier MUST resolve the agent's DID Document to obtain the public key:

1. Parse the DID to determine the method (`did:web`, `did:key`, etc.).
2. For `did:web`: HTTPS GET to `https://<domain>/.well-known/did.json`.
3. Locate the `verificationMethod` matching the proof's `verificationMethod` URL.
4. Decode the `publicKeyMultibase` value.
5. Cache the resolved key with a configurable TTL (RECOMMENDED: 300 seconds).

8.4 Failure Modes

Verification MUST fail if any of the following conditions are met:

- Data Integrity proof verification fails (signature invalid).
- `validUntil` < current time − clock skew.
- `validFrom` > current time + clock skew.
- `credentialSubject.intent.resource` is missing or empty.
- `credentialSubject.vouchVersion` is not a supported version.
- DID resolution fails and no trusted root is configured for the issuer.
- `credentialStatus` indicates revocation.

---

9. Delegation Chains

9.1 Overview

In multi-agent systems, a root principal may delegate authority through a chain of agents. The Vouch Protocol supports cryptographic delegation chains that preserve accountability at each hop. The semantics are aligned with W3C Authorization Capabilities (ZCAP-LD) [ZCAP-LD], with two design distinctions:

- **No JSON-LD requirement**: Delegation links use the same JCS-canonicalized JSON form as the surrounding credential, avoiding JSON-LD canonicalization.
- **Explicit resource binding**: Each delegation link MUST carry a `resource` field, ensuring that capabilities are bound to specific URIs rather than abstract action names.

9.2 Delegation Link Structure

Each link in a delegation chain contains:

```
{
  "issuer": "did:web:alice.example.com",
  "subject": "did:web:travel-agent.example.com",
  "intent": {
    "action": "plan_trip",
    "target": "destination:Paris",
    "resource": "https://travel-api.example.com/v1/bookings"
  },
  "validFrom": "2026-04-26T09:00:00Z",
  "validUntil": "2026-04-26T11:00:00Z",
  "proof": { ...DataIntegrityProof... }
}
```

Field	Required	Description
`issuer`	REQUIRED	DID of the delegator
`subject`	REQUIRED	DID of the delegate (recipient of authority)
`intent`	REQUIRED	Object describing the authorized scope, including required `resource`
`validFrom`, `validUntil`	REQUIRED	Temporal bounds for the delegation
`proof`	REQUIRED	Data Integrity proof signed by the `issuer`

9.3 Chain Validation

To verify a delegation chain, the verifier:

1. Validates the outermost Vouch Credential signature.
2. Walks the chain from the last link to the first.
3. For each link, verifies that `subject` matches the `issuer` of the next link.
4. Verifies the root `issuer` is a trusted principal.
5. Verifies that each downstream link's `intent.resource` is a sub-resource of, or equal to, the parent link's `intent.resource` (resource-narrowing rule).
6. Verifies that each link's temporal bounds are within the parent's bounds.

9.4 Depth Limit

Implementations MUST enforce a maximum chain depth. The RECOMMENDED maximum is 5 hops. See [PAD-022] Swarm Limits Protocol for handling of multi-agent delegation graphs that exceed simple linear chains.

9.5 Inverse Capability Pattern

For agents that require dynamic, attenuated authority, where a parent agent grants a sub-agent a reduced subset of its own capabilities, implementations SHOULD follow the Inverse Capability Pattern described in [PAD-021]. This pattern ensures that delegation links can only narrow, never broaden, the parent's authority.

---

10. Identity Sidecar Pattern

10.1 Motivation

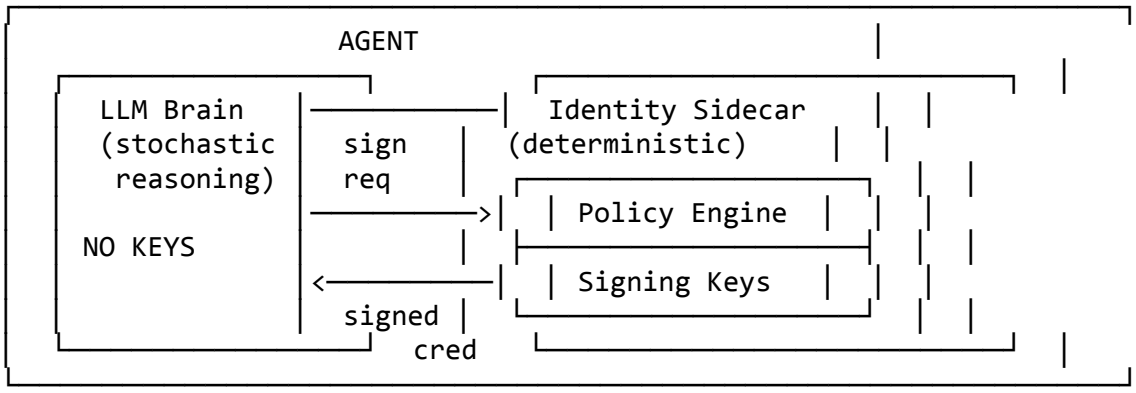
Large Language Models (LLMs) are non-deterministic and vulnerable to prompt injection attacks. Exposing private keys to an LLM's context window creates critical security risks:

- The LLM may leak the key in its output (via prompt injection).
- A jailbroken LLM may use the key without authorization.
- The key may persist in training data or logs.

10.2 Architecture

The Identity Sidecar pattern (described in defensive disclosure [PAD-003]) separates the agent into two processes:

1. **The Brain (Stochastic)**: The LLM that reasons and plans. It holds ZERO cryptographic secrets.
2. **The Passport (Deterministic)**: A sidecar process that holds private keys in secure memory and performs signing operations.



10.3 Just-In-Time (JIT) Signing Flow

1. The LLM decides to perform an action.
2. The LLM sends a structured signing request to the Sidecar with the intent payload.
3. The Sidecar evaluates the payload against a deterministic policy engine.
4. If the policy passes, the Sidecar issues the Vouch Credential.
5. The LLM attaches the credential to its API request.

If the policy check fails, the Sidecar MUST refuse to sign and MUST return an error explaining which policy was violated.

The deterministic policy engine at step 3 SHOULD enforce a **pre-declared intent allow-list** as part of its check: the set of `intent.action` values the Sidecar is configured to sign, evaluated as a structural predicate that does not consult the LLM. This pattern is described in [PAD-056] (Capability-Bounded AI Assistant Output via Intent Allow-List at the Identity Sidecar). It provides a structural capability bound that is independent of LLM behavior: even a fully prompt-injected or jailbroken LLM cannot cause the Sidecar to issue a credential for an `intent.action` outside the allow-list, because the allow-list is part of the Sidecar's deployment configuration and is not present in the LLM's prompt context. The allow-list bounds *capability type*, not *abuse within type*; within-allow-list misuse remains addressable by orthogonal mechanisms (rate-limiting, pattern strictness on `intent.target` and `intent.resource`, downstream verifier policy, and human-in-the-loop confirmation for irreversible actions).

10.4 MCP Integration

When used with the Model Context Protocol (MCP), the Identity Sidecar is exposed as an MCP Tool:

```
{
  "name": "vouch_sign",
  "description": "Issue a Vouch Credential for an intent payload",
  "inputSchema": {
    "type": "object",
    "properties": {
      "intent": {
        "type": "object",
        "description": "The action, target, and resource to sign",
        "required": ["action", "target", "resource"]
      }
    },
    "required": ["intent"]
  }
}
```

> **Editor's note (informative).** This subsection deliberately specifies only the minimal MCP tool surface needed to invoke the Sidecar. The richer binding

shipped by the reference implementation is documented immediately below as §10.4.1 (informative). Both will continue to evolve in step with the MCP specification.

10.4.1 Reference MCP Binding (Informative)

The reference implementation under `vouch.integrations.mcp` exposes three MCP tools and a small amount of MCP-resource surface. Implementations MAY adopt this shape directly; it is informative for v1.0 and is expected to firm up as the MCP specification stabilizes.

Tools. Three tools are registered, each addressing a distinct lifecycle moment:

Tool name	Purpose	Returns
`sign_action`	Mint a single short-lived Vouch Credential bound to one `intent` (`action` + `target` + `resource`). Default credential lifetime 300 s.	Signed credential as a JSON object; or compact JWS string for legacy callers.
`create_session`	Mint a longer-lived session credential covering multiple actions whose scope is declared up front. Used when per-action signing is too chatty (e.g. interactive agent loops).	Session credential; the agent attaches it as a parent credential on subsequent `sign_action` calls to form a delegation chain.
`get_status`	Return the issuer DID, the cryptosuite in use, the configured `validUntil` ceiling, and whether the Sidecar is in auto-sign mode.	Plain JSON status object; not signed.

Auto-sign mode (informative). The reference Sidecar honors a `VOUCH_AUTO_SIGN` environment variable. When unset, the host LLM agent MUST receive an explicit `sign_action` tool call from the planner. When set to a truthy value, the Sidecar's deterministic policy engine (Section 10.3) MAY auto-sign for `intent.action` values that are on the pre-declared allow-list (`PAD-056`). This is a deployment-time choice; the credential format is identical either way.

Error codes. The reference binding emits structured MCP errors using a `vouch.error.*` namespace so a client can branch on cause rather than parsing prose. Five codes are stable: `vouch.error.no_signer_configured`, `vouch.error.intent_outside_allowlist`, `vouch.error.delegation_resource_mismatch`, `vouch.error.heartbeat_required`,

`vouch.error.cryptosuite_unavailable`. These are documented in `vouch.integrations.mcp.errors`; new codes may be added in v0.2.

DID Document as MCP resource. The reference server registers the agent's `did.json` as a fetchable MCP resource at URI `vouch://did/{did}/document`, so a verifier-side MCP client can resolve the issuer DID over the same channel it called the signing tool, without a separate HTTPS fetch. This is an MCP-native alternative to the `did:web` HTTPS resolution path in Section 6.

Why this is informative. The MCP specification was in active revision at the time of this report. Pinning a normative binding now risks a versioning mismatch within months of publication. The format above represents the shape that has converged across the Python, TypeScript, and Go reference implementations and is stable enough to copy; the W3C-CCG submission will revisit normativity for v0.2 once the MCP specification has stabilized.

10.5 Algorithm Migration

The Sidecar boundary makes cryptographic algorithm migration drop-in. The LLM never sees algorithm-specific material; transitioning from Ed25519 to the dual-proof post-quantum profile (Section 13) requires no LLM-side changes.

10.6 Reference Sidecar Tiering (Informative)

This subsection is informative implementation guidance, not normative. It captures the lessons from the three reference sidecars maintained alongside this specification.

Three reference sidecar implementations ship with this specification. They are intentionally **not** feature-equivalent. Implementers selecting a sidecar for production deployment SHOULD pick by tier, not by language preference.

Tier	Reference implementation	Key storage	Typical deployment
Production	Go (`go-sidecar/`)	KMS / HSM / file	Audited, regulated, or high-throughput
Lightweight	Python (`vouch.sidecar.*`)	File or env	Non-regulated self-hosted, Python-native stacks
Lightweight	TypeScript (`packages/sdk-ts/sidecar/`)	File or env	Non-regulated self-hosted, Node-native stacks
Development	Python `dev_sidecar` (`website-agent/backend/vouch_agent/dev_sidecar.py`)	Ephemeral, in-memory	Local development, demos

The **Lightweight** tier deliberately omits features available in the Production tier:

- Dual-proof post-quantum signing (Section 13)
- KMS / HSM key integration
- Sensitive-mode JWE wrapping
- Heartbeat session validation (Section 11)
- Multi-tenancy

When an implementation requires any of these, deployers MUST migrate to the Production tier sidecar. The HTTP wire contract is identical across tiers, so this migration is a single environment-variable change at the client.

All three sidecars expose the following HTTP API:

- `GET /health`: liveness probe
- `GET /did`: the sidecar's configured DID
- `GET /.well-known/did.json`: the DID Document (optional, dev-friendly)
- `POST /sign`: sign a Vouch Credential for an intent

A shared **contract test suite** (`test-vectors/sidecar-contract/`) verifies that each implementation accepts and rejects the same inputs and emits semantically equivalent credentials. Implementers building a fourth sidecar (in another language or another runtime) SHOULD use this contract suite for conformance.

This tiering is informative and does not impose normative requirements on implementations. An implementer MAY provide a single feature-complete sidecar in any language; the tier structure is a guidance pattern, not a conformance requirement.

---

11. Credential Lifecycle and the Heartbeat Protocol

11.1 The Heartbeat Protocol

For long-running agents, the Vouch Protocol defines a continuous credential renewal mechanism, the Heartbeat Protocol. It inverts the traditional PKI trust model from "Trusted until Revoked" to **"Untrusted until Renewed"**. The Heartbeat Protocol is described in defensive disclosure [PAD-016] (Dynamic Credential Renewal) and complemented by [PAD-020] (Ratchet Lock Protocol) for cryptographic continuity. For high-stakes agent actions where minimum elapsed wall-clock time between credential issuances must be cryptographically self-evident without trust in any clock authority, [PAD-047] (Verifiable Delay Functions for Cryptographic Rate-Limiting) describes a complementary rate-limiting primitive that uses VDF outputs as proof-of-elapsed-time.

The Heartbeat Protocol shares conceptual ground with two adjacent W3C-CCG work items:

- **Cryptographic Event Logs (cel-spec) and did:cel.** Like CEL, a Vouch heartbeat chain is an append-only sequence of hash-linked entries that any verifier can replay deterministically. Vouch heartbeats differ in that each entry carries an explicit behavioral attestation payload (counters, anomaly votes, ratchet state) over and above the link hash, and that the chain terminates a per-session trust budget rather than a per-identity history. The two formats are not mutually exclusive: a Vouch heartbeat chain MAY be serialized as a CEL stream, and a verifier accepting CEL entries can validate a Vouch chain by treating heartbeat payloads as CEL event data.
- **VCALM (Verifiable Credential API for Lifecycle Management).** The Heartbeat Protocol is not a credential-lifecycle-management API. It is an in-session renewal mechanism that produces short-lived Session Vouchers from a longer-lived issuer credential, with behavioral attestation gating each renewal. VCALM-style operations (issuance, revocation, suspension, status updates against a registry) are out of scope for the Heartbeat Protocol and SHOULD be carried by the surrounding VCALM-compatible service when one is present.

11.2 Credential Status

Vouch Credentials MAY include a `credentialStatus` property referencing a BitstringStatusList [VC-BITSTRING-STATUS-LIST] for credential-level revocation or suspension. Implementations SHOULD cache status lists locally and refresh them on verification failure.

A reference implementation of both the issuer and verifier sides of BitstringStatusList ships across the Python, TypeScript, and Go SDKs. The three implementations share a single cross-language test vector at `test-vectors/bitstring-status-list/vector.json` and produce equivalent decoded bitstrings (Python and TypeScript produce byte-identical encoded output; Go's `compress/flate` produces a valid DEFLATE stream that decodes to the same bitstring, the equivalence W3C requires). Each implementation provides:

- A `StatusList` class encapsulating the W3C-compliant gzip + base64url multibase bitstring of minimum length 131,072 bits per W3C §4.2, with deterministic gzip headers (mtime fixed to 0, OS field normalized to 0xff) so issuers can publish reproducible status credentials.
- A `BitstringStatusListCredential` builder that produces an unsigned credential ready for Data Integrity proof attachment and publication at a stable URL.
- A `credentialStatus` entry builder that constructs the W3C-shaped reference attached to a Vouch Credential, with index, purpose, and structural validation.
- A `verify_status` function that decodes a fetched status list credential, validates structural fields (id, type, purpose, encoded list presence), and returns the bit value for the credential's index.
- A persistence path via `to_state_dict` / `from_state_dict`, carrying the encoded bitstring and the allocation cursor (which is NOT recoverable from the encoded

list alone) so issuers can survive restarts without re-allocating already-used indices. A reference `FilesystemStatusListStore` ships in the Python SDK; production deployments substitute Redis, Postgres, S3, or equivalent.

- An HTTP fetcher (`vouch.status_list_fetcher.StatusListFetcher` in the Python SDK) with an in-memory TTL cache, conditional GETs via `ETag` / `If-Modified-Since`, a configurable response-size limit, and a `force_refresh` parameter that verifiers SHOULD set on verification failure to handle stale-cache scenarios. TypeScript and Go callers can compose the equivalent pattern with the platform's built-in HTTP client (`fetch`, `net/http`).

Complementary to credential-level status, the Python SDK additionally ships an informative DID-level revocation registry (`vouch.revocation`) with pluggable Memory, Redis, and HTTP-remote backends. This is useful for operators that need to revoke all credentials issued under a compromised agent identity in a single operation, distinct from credential-by-credential status updates. Implementers MAY use the BitstringStatusList path, the DID-level registry, both in combination, or substitute their own.

11.3 Heartbeat Request

Every T seconds (where T < `validUntil` − `validFrom`), the agent submits a heartbeat request:

```
{
  "version": "1.0",
  "type": "heartbeat_request",
  "agentDid": "did:key:z6MkAgent...",
  "sequenceNumber": 4207,
  "timestamp": "2026-04-26T10:00:00Z",
  "nonce": "<random-32-bytes-hex>",
  "previousVoucherHash": "sha256:<hash>",
  "actionMerkleRoot": "sha256:<merkle-root-of-actions>",
  "behavioralDigest": {
    "apiCalls": 47,
    "tokensConsumed": 12400,
    "resourcesAccessed": ["database:read", "api:weather"],
    "intentDriftScore": 0.02
  },
  "canaryReveal": "<previous-secret-plaintext>",
  "canaryCommitment": "sha256:<hash-of-new-secret>",
  "proof": { ...DataIntegrityProof... }
}
```

11.4 Session Voucher

Upon successful validation, the Validator quorum issues a Session Voucher (a Verifiable Credential of type `SessionVoucher`):

```
{
  "@context": ["https://www.w3.org/ns/credentials/v2",
"https://vouch-protocol.com/contexts/v1"],
  "type": ["VerifiableCredential", "SessionVoucher"],
  "credentialSubject": {
    "id": "did:key:z6MkAgent...",
    "decayLambda": 0.05,
    "initialTrust": 1.0,
    "maxTtl": 60,
    "scope": ["api:read", "api:write", "database:read"]
  },
  "issuer": ["did:web:validator-a", "did:web:validator-b",
"did:web:validator-c"],
  "validFrom": "2026-04-26T10:00:00Z",
  "validUntil": "2026-04-26T10:01:00Z",
  "proof": [ {...}, {...}, {...} ]
}
```

11.5 Trust Entropy

Trust decays continuously via an exponential function:

```
trust(t) = initialTrust * e^(-decayLambda * (t - issuedAt))
```

Verifiers apply operation-specific thresholds:

Operation Category	Trust Threshold	Approximate Recency
Financial transaction	0.95	Within ~1 second
API write	0.80	Within ~4 seconds
API read	0.50	Within ~14 seconds
Health check	0.20	Within ~32 seconds
Logging	0.05	Within ~60 seconds

Trust Entropy is described as a defensive disclosure in [PAD-030] (ZK Reputation Portability) and [PAD-036] (Aggregated Reputation Scoring). It provides a mathematical, deterministic alternative to subjective reputation scoring, verifiers compute trust by formula rather than consulting a third-party reputation service.

11.6 Federated Validator Quorum

Renewal requires independent M-of-N approval from Validators evaluating different safety dimensions:

- **Policy Validator**: Governance rules, regulatory compliance, kill switches.
- **Behavioral Validator**: Intent drift detection, anomaly detection.
- **Budget Validator**: Resource consumption limits, cost ceilings.

Each Validator signs independently with its own key. No single Validator compromise can issue valid Session Vouchers. The federation pattern is described in [PAD-037] (Credential Federation).

11.7 Canary Commitments and Cryptographic Mortality

The Heartbeat Protocol incorporates a decentralized dead-man's-switch mechanism:

1. Each heartbeat **reveals** the previous interval's secret (proving agent continuity).
2. Each heartbeat **commits** to a new secret (enabling future death detection).
3. If the agent fails or is substituted, the unrevealed secret signals failure to any verifier.

This pattern, combined with the Cryptographic Mortality Protocol described in [PAD-032], enables verifiers to distinguish between an agent that is silent (legitimately idle) and an agent that is dead or substituted (cryptographic continuity broken).

11.8 Adaptive TTL

TTL is dynamically computed based on trust history:

Agent State	Clean Heartbeats	TTL
New / Unknown	0	5 seconds
Probationary	50	~10 seconds
Established	200	~18 seconds
Trusted	500	~28 seconds
Veteran	2000	~45 seconds
Anomaly Detected	Any	Instant reset to 5 seconds

---

12. Security Considerations

This section is structured as a STRIDE-shaped threat model per the W3C Security Interest Group's Threat Modeling Guide. Once implementer interest broadens and operational deployments accumulate, this section and Section 14 (Privacy Considerations, LINDDUN-shaped) are expected to be factored into a companion *Vouch Protocol Threat Model* document.

Assets in scope: the issuer's private signing key; the integrity of the `intent` payload (action, target, resource); the authenticity of the issuing agent's DID; the audit chain produced by the Heartbeat Protocol; the verifier's correct accept/reject decision.

Adversary capabilities assumed: an active network attacker (read, drop, modify, replay traffic); a fully prompt-injected or jailbroken LLM process; a compromised

individual verification key; a malicious credential issuer attempting to impersonate another agent; a colluding intermediate validator in a heartbeat federation (Section 11.6). Out of scope: physical extraction of HSM-resident keys; attacks on the underlying Ed25519 or ML-DSA-44 primitives themselves; supply-chain compromise of the verifier binary.

12.1 Spoofing

Threat	Mitigation
Attacker presents a credential claiming to be issued by another agent's DID.	The credential is signed by an Ed25519 / ML-DSA-44 key whose public counterpart is published in the issuer's DID Document (Sections 4, 6). A verifier MUST resolve the DID, fetch the Multikey, and reject any credential whose signature does not validate against a `verificationMethod` listed under the DID.
Attacker registers a look-alike domain (e.g. `did:web:agent.acme.example` with Cyrillic "a") and serves a DID Document.	Verifier policy (out of scope of this specification) SHOULD apply IDN-homograph checks before trusting a newly-seen DID. Vouch does not address domain-name infrastructure trust directly.
Prompt-injected LLM impersonates the agent to a downstream API by inventing a token.	The signing key lives in the Identity Sidecar (Section 10), never in the LLM context window. A jailbroken LLM cannot mint a credential because it does not hold the key.

12.2 Tampering

Threat	Mitigation
Attacker modifies the `intent` payload of a signed credential in flight.	The Data Integrity proof (`eddsa-jcs-2022`) covers the JCS-canonicalized credential bytes (Section 7). Any modification of a covered field invalidates the proof.
Attacker replays a previously-issued credential against a different resource.	The REQUIRED `intent.resource` field binds the credential to a specific resource URI (Section 5.4.1). Implementations MUST reject credentials where `resource` is missing, empty, or syntactically invalid as a URI.
Attacker replays an unmodified credential to the same resource within its validity window.	Vouch Credentials carry `validFrom` and `validUntil` (default 300 s window) and a RECOMMENDED `id`. Verifiers SHOULD maintain a cache of recently seen

Threat	Mitigation
	credential ids to detect replay attempts within the validity window. The cache MUST persist across verifier restarts to maintain the guarantee during operator-initiated downtime.
Attacker downgrades the cryptosuite to a weaker algorithm.	The cryptosuite is declared explicitly in `proof.cryptosuite`. There is no negotiation surface; a verifier configured to require ML-DSA-44 (per the dual-proof policy of Section 13.2 Mode B or C) MUST reject any credential whose `proof` array does not include a valid `mldsa44-jcs-2026` proof.

12.3 Repudiation

Vouch Credentials are non-repudiable by design: each carries a Data Integrity proof signed by the agent's key, the canonical bytes are reproducible by any party, and the proof binds the issuer's DID to the asserted `intent` at the asserted `created` timestamp. An issuer that later disputes "I signed credential X" can be confronted with the bytes plus the resolvable public key from the DID Document. (The privacy tension this creates for human principals is addressed under LINDDUN in Section 14.4.)

12.4 Information Disclosure

Threat	Mitigation
Prompt injection causes the LLM to leak the signing key.	The Identity Sidecar (Section 10) physically isolates the key. The LLM emits a tool-call object; the orchestration layer asks the Sidecar to sign; the Sidecar returns a credential bound to that action. The key is never in the LLM's context window, so it cannot leak through prompt injection.
Intent payload contains regulated data (PHI, financial routing) and is visible to any party on the wire.	Vouch Credentials in v1.0 are non-confidential. Implementers transmitting sensitive intents SHOULD use a confidentiality layer (TLS at minimum) and SHOULD minimize what they put into the intent payload (see Section 14.5 for the LINDDUN privacy frame). A future OPTIONAL confidentiality profile using post-quantum key encapsulation is anticipated.

Threat	Mitigation
Reasoning unfaithful to retrieved context (RAG hallucination) leads to a signed credential for an action the agent should not have taken.	[PAD-045] describes a mechanism for cryptographically anchoring the agent's reasoning to specific retrieved documents, so downstream verifiers can detect whether the agent's output diverged from its retrieval. Informative for v1.0.
Model-weight substitution or unauthorized fine-tune produces a different agent under the same DID.	[PAD-043] describes a mechanism for binding the agent's DID to a fingerprint of its model weights at issuance time. Informative for v1.0.

12.5 Denial of Service

Threat	Mitigation
Attacker floods the Sidecar with sign requests, exhausting the signing key's rate budget or the Sidecar's compute.	The Sidecar's deterministic policy engine (Section 10.3) SHOULD enforce per-source and per-action rate limits. [PAD-047] describes a verifiable-delay-function pattern for cryptographically rate-limited issuance where wall-clock guarantees matter.
Attacker submits credentials with extremely long `proof.proofValue` or pathological JCS-canonical inputs to exhaust the verifier.	Verifiers SHOULD bound the maximum credential size they accept (a 32 KB upper limit is sufficient for the dual-proof profile with a 4-entry delegation chain) and SHOULD enforce a per-source verification-rate limit.
Attacker provokes the verifier into expensive DID-resolution storms by submitting credentials referencing many distinct, unresolvable DIDs.	Verifiers SHOULD cache DID resolutions with a bounded LRU and fail-fast for resolution failures rather than retrying inline.

12.6 Elevation of Privilege

Threat	Mitigation
Issuer's private key is compromised; attacker mints credentials with the agent's full authority.	(a) The DID Document MUST be updated to remove or rotate the compromised key, immediately invalidating credentials signed under it for verifiers that re-resolve. (b) Under the Heartbeat Protocol (Section 11), credentials expire naturally within `maxTtl` (default 60 s). (c) Canary commitments (Section 11.7) enable detection of agent

Threat	Mitigation
	substitution by exposing an unrevealed secret. (d) The credential's `credentialStatus` MAY be set to revoked via the BitstringStatusList mechanism for an immediate signal to caching verifiers.
Jailbroken LLM crafts an `intent` outside its allow-listed action vocabulary.	The Sidecar's deterministic policy engine enforces a pre-declared intent allow-list (PAD-056) as a structural predicate that does not consult the LLM. Even a fully-jailbroken LLM cannot cause the Sidecar to sign for an `intent.action` outside the allow-list.
Attacker uses a credential issued for one resource to access a related, more-privileged resource (confused deputy).	Resource binding via the REQUIRED `intent.resource` field (Section 5.4.1). Verifiers MUST compare the request's effective resource URI against `intent.resource` and reject when they do not match.
Compromised intermediate validator in a heartbeat federation issues unauthorized Session Vouchers.	Each Validator signs independently with its own key. Quorum policy (Section 11.6) is M-of-N; no single Validator compromise can produce a valid Session Voucher.

12.7 Foundational Cryptographic Choices

The threat model above rests on a small set of primitive choices. Each is summarized here for traceability:

Component	Algorithm	Rationale
Default signature	Ed25519 [RFC 8032]	Modern, fast, small signatures, no known weaknesses
Dual-proof signature (optional)	Ed25519 + ML-DSA-44 [FIPS 204]	Quantum-safe migration path; see Section 13
Cryptosuite	`eddsa-jcs-2022` [VC-DI-EDDSA]	W3C Data Integrity standard with JCS canonicalization
Canonicalization	JCS [RFC 8785]	Deterministic, parser-independent, no JSON-LD overhead
Key encoding	Multikey [W3C Controlled Identifiers]	Algorithm-agnostic, future-compatible

Component	Algorithm	Rationale
Key encryption	ChaCha20-Poly1305	AEAD cipher, timing attack resistant
Key derivation	Scrypt	Memory-hard, GPU/ASIC resistant

---

13. Crypto-Agility and Quantum-Safe Profile

13.1 Motivation

Several regulatory frameworks, including U.S. National Security Memorandum 10, NIST CNSA 2.0, and CNSSP-15, require migration to quantum-resistant cryptography on defined timelines. The “harvest-now, decrypt-later” threat model is recognized: while Vouch Credentials are short-lived (default 5 minutes), retroactive forgery of historical credentials in a post-quantum era could undermine non-repudiation guarantees relied upon for audit, regulatory submissions, and liability determinations.

13.2 Dual-Proof Post-Quantum Profile (Optional)

Implementations MAY pair a classical signature with a post-quantum signature on the same Vouch Credential by attaching **two independent Data Integrity proofs** to the credential, rather than minting a bespoke composite cryptosuite. Each proof is a standalone, standards-aligned Data Integrity proof; together they provide post-quantum resilience while remaining verifiable by any verifier that understands either cryptosuite.

The dual-proof profile is defined as follows:

- The agent’s DID Document publishes an Ed25519 Multikey and an ML-DSA-44 [FIPS 204] Multikey under separate `verificationMethod` slots.
- The credential’s `proof` field is an array containing two Data Integrity proof objects:
  - One proof with `cryptosuite: "eddsa-jcs-2022"` (the default classical cryptosuite of Section 7).
  - One proof with `cryptosuite: "mldsa44-jcs-2026"` (the post-quantum cryptosuite; see Section 13.5 for its provisional identifier and ongoing coordination with the W3C Data Integrity Working Group and the Digital Bazaar `mldsa44-rdfc-2024-cryptosuite` family).
- Both proofs are computed over the **same** RFC 8785 JCS-canonicalized credential bytes. This shared-bytes property is preserved by construction because both cryptosuites use JCS canonicalization, and is documented as PAD-040 (Dual-Proof Same-Canonical-Bytes Property).
- Verifier policy decides which proofs must validate:
  - **Mode A (classical-only):** validate the `eddsa-jcs-2022` proof. Useful for verifiers that have not yet been upgraded to ML-DSA-44.

- **Mode B (post-quantum-only):** validate the `mldsa44-jcs-2026` proof. Useful for verifiers operating under strict PQ migration mandates.
- **Mode C (both required, recommended for regulated audit credentials):** validate both proofs. The credential is accepted only if every proof in the array verifies.

The verifier policy is local to the verifier and is not embedded in the credential itself, which keeps the wire format identical regardless of the verifier's posture.

Why dual proofs rather than a composite cryptosuite. An earlier revision of this specification defined a single composite cryptosuite (`hybrid-eddsa-mldsa44-jcs-2026`) whose `proofValue` was a concatenation of an Ed25519 signature and an ML-DSA-44 signature. The dual-proof formulation is preferred because (a) each proof uses a separately-standardized cryptosuite identifier, avoiding the need to register a Vouch-specific composite identifier; (b) the Data Integrity `proof` field is already specified as an array, so dual-signing is a natural use of existing primitives; (c) future expansion to additional cryptosuites (ML-DSA-65, SLH-DSA, hash-based schemes) becomes additive rather than requiring a new composite identifier; and (d) verifiers that understand only one cryptosuite remain interoperable without needing to parse a bespoke composite proofValue. The composite cryptosuite identifier `hybrid-eddsa-mldsa44-jcs-2026` is retained as a transitional alias for backward compatibility with v1.6.x reference implementations; new implementations SHOULD emit dual proofs.

13.3 Migration Guidance

The conformance level of the dual-proof profile is expected to evolve in step with external post-quantum migration milestones (NIST CNSA 2.0 phased adoption, NSM-10 obligations, CNSSP-15 timelines), rather than fixed Vouch-internal dates:

- **This revision:** `eddsa-jcs-2022` is the default conformance cryptosuite. Pairing it with an additional `mldsa44-jcs-2026` Data Integrity proof on the same credential is OPTIONAL (MAY).
- **As CNSA 2.0 phase-in advances and regulator guidance matures:** the dual-proof profile is expected to be RECOMMENDED (SHOULD) for deployments in regulated sectors.
- **As classical-only signatures reach end-of-life under those policies:** a dual-proof or pure-PQ profile is expected to be REQUIRED (MUST) for new deployments.

Implementers operating in regulated sectors are advised to align dual-proof adoption with the post-quantum migration schedule applicable to them.

13.4 Payload Considerations

ML-DSA-44 produces signatures of approximately 2,420 bytes and public keys of approximately 1,312 bytes. A credential carrying both an `eddsa-jcs-2022` proof and an

`mldsa44-jcs-2026` proof therefore exceeds typical HTTP header size limits. Implementations using the dual-proof profile SHOULD transmit credentials in the HTTP request body (Section 5.6).

The defensive disclosure portfolio includes additional optimizations for post-quantum payloads: - [PAD-033] ZK PQ Signature Compression (zero-knowledge proof compression of post-quantum signatures). - [PAD-034] Composite Threshold Swarm Consensus (BLS aggregation paired with ML-DSA for multi-agent authorization). - [PAD-035] Async Chunked Edge PQ Signatures (chunked verification of large hash-based signatures).

These are referenced as informative prior art and are not normative requirements of v1.0.

13.5 Algorithm Identifiers

Algorithm	Multicodec	Cryptosuite Identifier
Ed25519	`ed25519-pub` (`0xed01`), registered	`eddsa-jcs-2022`
ML-DSA-44	`mldsa44-pub`, registered (see the multicodec table for the assigned code)	`mldsa44-jcs-2026` (provisional; to be aligned with the W3C Data Integrity Working Group's selection, expected to converge with the Digital Bazaar `mldsa44-rdfc-2024-cryptosuite` family's forthcoming JCS variant)

The dual-proof profile of Section 13.2 uses two independent Multikey entries (one Ed25519, one ML-DSA-44) in the issuer's DID Document, each with its already-registered multicodec; no separate composite multicodec is defined. Final cryptosuite identifiers will be coordinated with the W3C Data Integrity Working Group and the multicodec registry.

13.6 Algorithm Quorum Verification (Optional)

During the post-quantum transition period, verifiers MAY require that a credential carry signatures from M-of-N independent cryptosuites (for example, Ed25519 plus ML-DSA-44 plus SLH-DSA-128s), so that compromise of any single algorithm family does not compromise the credential. Each signature attests to the same RFC 8785 JCS-canonicalized payload. The verifier accepts the credential only if at least M of the N signatures verify; the specific M-of-N policy is declared by the verifier, not embedded in the credential itself.

This pattern is described in [PAD-046] (Algorithm Quorum Verification via M-of-N Cryptosuite Diversity) and is non-normative for v1.0. Implementations that require

defense-in-depth against future algorithmic breaks SHOULD consider it. The dual-proof post-quantum profile of Section 13.2 is the simplest 2-of-2 case (Ed25519 + ML-DSA-44 both required, expressed as two Data Integrity proofs on the same credential); algorithm quorum generalizes to arbitrary M-of-N configurations and to threshold tolerance for partial breaks.

13.7 Adjacent Work in the CCG

Two closely related developments from the W3C-CCG ecosystem are tracked here as forward-pointers and complement (rather than substitute for) the profile in this Section:

- **Bloom-filter publishing of cryptographic digests for key-compromise reporting.** A compact, append-only signal that a previously-issued verification key is now considered compromised, designed so verifiers can fail-closed against an entire compromised key without round-tripping to a status-list registry per credential. Vouch verifiers MAY consume such a signal (when standardized) as a pre-check before validating a Data Integrity proof; this is complementary to the credential-level `BitstringStatusList` path in Section 11.2.
- **Post-quantum protection over pre-quantum signatures.** Schemes that bind a classical Ed25519 (or ECDSA) signature to a post-quantum commitment, so that a classical-only credential issued today can still be evidenced as PQ-safe at verification time once the commitment registry is available. Vouch is interested in this pattern as an alternative compatibility path for issuers that cannot yet emit a full dual-proof credential; tracking the W3C-CCG/Digital Bazaar work in this area is a stated goal.

These items are informative for v1.0 and will be re-examined when their specifications mature.

---

14. Privacy Considerations

This section is structured as a LINDDUN-shaped threat model, the privacy counterpart to STRIDE for Section 12. The categories below are: **L**inkability, **I**dentifiability, **N**on-repudiation (the privacy reading), **D**etectability, **D**isclosure of Information, **U**nawareness, and **N**on-compliance.

Privacy assets in scope: the human or organizational principal behind an agent's DID; the relationship between distinct credentials issued by the same agent; the contents of `intent` payloads; the membership of delegation chains; the behavioral attestation payloads in heartbeats. **Data subjects:** human end-users whose data the agent acts upon; principals whose authority the agent invokes; operators whose infrastructure issues credentials.

14.1 Linkability

A persistent agent DID lets a single verifier (or two colluding verifiers) link multiple credentials as having come from the same agent. This is inherent to public-key cryptography: the same public key validating two signatures *means* the same private key produced both.

Mitigations available to implementers:

- Privacy-sensitive deployments MAY rotate agent DIDs on a schedule with no cross-DID linkage, accepting the audit-chain discontinuity as the trade-off.
- `did:key` (rather than `did:web`) avoids tying the agent identifier to a domain that may also host other services.
- A future revision MAY incorporate blind-witness commitments in heartbeat chains so an external auditor can attest "the agent kept renewing trust" without learning what the agent did. This is an open research direction informed by the Cryptographic Event Log work cited in Section 11.1.

14.2 Identifiability

The agent DID resolves to a DID Document which, for `did:web`, is published at a domain that may identify a real organization. The delegation chain (Section 9) further reveals the human or organizational principal who delegated authority. A passive observer of the public credential bytes can identify the parties involved at each link of the chain.

Implementations MAY support selective disclosure of delegation chains, omitting intermediate links if the verifier only needs to verify the root and leaf. Issuers SHOULD avoid populating `intent.target` and `intent.resource` with values that themselves identify a third party (e.g. a specific patient identifier) when a less-identifying value (a record-class identifier) suffices.

14.3 Non-repudiation (privacy reading)

Vouch Credentials are non-repudiable by design (Section 12.3 covers the security reading). The privacy reading is that this design choice removes plausible deniability from the human principal whose key signed the credential. In contexts where the principal would prefer the ability to deny having authorized an action (e.g. whistleblower-adjacent scenarios), the cryptographic non-repudiation cuts against them.

Vouch is intentionally non-repudiable; this is a feature for the regulated-workload audience the specification primarily targets. Principals who need deniability SHOULD use a different identity layer (such as unlinkable group signatures) for the actions in question. Vouch is not the right tool for that case.

14.4 Detectability

A passive observer who sees the HTTP request body containing a Vouch Credential can detect that an agent action took place at that time, even without being able to read the `intent` payload (under a hypothetical confidentiality layer). The size and shape of the credential, the destination host, and the timing leak information.

Where action concealment matters, implementers can:

- Pad credentials to a fixed size class so the wire-shape does not reveal which `action` was invoked.
- Route traffic through a mixed-traffic network layer (Tor, Oxen, mixnets) when concealment matters more than performance.
- Emit decoy heartbeats during legitimately idle periods so an observer cannot distinguish silence-from-inactivity from silence-from-incapacitation.

14.5 Disclosure of Information

Vouch Credentials in the v1.0 profile are non-confidential: the `intent` payload is readable by any party that can read the credential. For sensitive intent payloads (PHI, financial routing instructions, proprietary corporate data), this is a privacy concern.

Always transmit credentials over a confidentiality-providing transport: TLS 1.3 with strong cipher suites at minimum. Minimize the information in `intent.target` and `intent.resource`; opaque record handles are preferred over human-readable identifiers when the verifier only needs to confirm scope. A future OPTIONAL confidentiality profile for `intent` using post-quantum key encapsulation is anticipated, but the mechanism is intentionally not specified in this revision.

14.6 Unawareness

Human principals whose authority an agent exercises may not be aware, at the moment of any specific action, that a Vouch Credential is being issued in their name. This is unlike a click-through consent flow where the human is in the loop on each action.

Deployments SHOULD provide principals with an audit interface (a “what has been signed on my behalf” view) and SHOULD log heartbeat-renewal events to that interface. For high-impact actions (irreversible transactions, deletions, regulated disclosures), implementations SHOULD require an explicit human-in-the-loop confirmation distinct from the agent’s automated allow-list. Behavioral digests in heartbeat requests SHOULD be aggregated (rather than individual action logs) when used for outbound reporting that the principal cannot opt out of.

14.7 Non-compliance

Issuing Vouch Credentials creates personal data and audit-log data subject to GDPR, HIPAA, the EU AI Act, and analogous regimes. Verifiers and Sidecars that retain

credentials must comply with retention, access, and erasure requirements in the relevant jurisdiction.

Implementer obligations under these regimes:

- Verifiers SHOULD NOT store Vouch Credentials beyond the retention period needed for audit purposes, and SHOULD support a credential-erasure path keyed by `credential.id` for in-jurisdiction subject-access requests.
- Issuers SHOULD document, per regulated workload, which legal basis under GDPR Article 6 (or the equivalent in the applicable regime) the credential issuance relies on, and SHOULD make that documentation discoverable through the principal's audit interface (Section 14.6).
- The companion `docs/compliance/*` collection maps Vouch primitives to the major regimes; implementers in regulated sectors should consult it.

---

15. State Verifiability (Informative)

> **Editor's note (informative).** Sections 15 and onward provide architectural context and implementer guidance. A subsequent revision is expected to factor this material into a separate companion document, leaving the present Report focused on the normative credential format, conformance, and the Heartbeat Protocol. The reader who wants only the conformance surface may stop at the end of Section 14; the reader who wants the operating model continues here.

This section is informative and describes the architectural layer that Vouch Protocol occupies relative to adjacent specifications.

Agent identity in 2026 is addressed by multiple specifications, including but not limited to:

- **Identity layer**: Decentralized Identifiers, Verifiable Credentials.
- **Authorization layer**: ZCAP-LD, OAuth 2.1 with mTLS.
- **State verifiability layer**: Vouch Protocol.

The State Verifiability layer answers operational questions that arise *after* an agent has been identified and authorized:

1. **Continuous attestation**: Is the agent's runtime state still aligned with its authorization, or has it drifted?
2. **Cryptographic continuity**: Has the agent been substituted, dead-keyed, or hijacked since its last verification?
3. **Behavioral provenance**: Can we cryptographically reconstruct what the agent did, in what order, with what resources?
4. **Quantum-safe non-repudiation**: Will the cryptographic proofs we accept today still bind the agent in 2030 and beyond?

5. **Federated trust evaluation**: Can multiple independent observers (policy, behavior, budget) agree on continued trust without a central authority?

Vouch Protocol provides primitives, Heartbeat, Trust Entropy, Validator Quorum, Canary Commitments, dual-proof post-quantum signatures, that compose with identity and authorization specifications. The protocol does not replace those specifications; it adds a continuous-state dimension beneath them.

Cross-implementation byte-identical canonicalization, achieved through RFC 8785 JSON Canonicalization, is what makes multi-party trust state in a heterogeneous Vouch deployment deterministically computable. Three independent implementations (Python, TypeScript, Go) of the same canonical form produce byte-identical credentials given the same input; this property is documented as [PAD-039] (JCS Deterministic Multi-Party Trust State). The metadata-schema conventions used by Vouch issuers across runtime variants, ensuring that `agent_runtime`, `model_provider`, and `agent_version` are populated consistently for downstream auditability, are documented as [PAD-042] (Standardized Metadata Schema for AI Agent Ledger Signatures).

For high-frequency agent-to-agent negotiation where on-chain or per-action credential issuance is too expensive, [PAD-044] (Ephemeral ZK-State Channels for Agentic Layer 2 Scalability) describes a layer-2-style channel pattern in which agents open a ZK-anchored state channel, exchange credentials off-channel, and settle a single proof on the trust ledger. This is informative in this revision and may be normatively integrated in a future revision.

A future revision of this specification is expected to make portions of the State Verifiability layer normative, including required behavioral attestation fields and validator quorum thresholds for regulated deployments, as implementer experience accumulates and regulatory expectations mature.

---

16. Implementer Interest and Charter Sponsors

16.1 Reference Implementations

This specification is accompanied by three open-source reference implementations of the protocol primitives:

- **Python**: `vouch-protocol` package on PyPI. Source at `https://github.com/vouch-protocol/vouch/tree/main/vouch/`. Includes the reference Signer, Verifier, BitstringStatusList, delegation chain validator, and the State Verifiability runtime (Section 15).
- **TypeScript**: `@vouch-protocol/core` on npm. Source at `https://github.com/vouch-protocol/vouch/tree/main/packages/sdk-ts/`. Includes the TypeScript Signer, Verifier, BitstringStatusList, and the dual-proof post-quantum profile.

- **Go**: `github.com/vouch-protocol/vouch/go-sidecar` on Go modules. Source at `https://github.com/vouch-protocol/vouch/tree/main/go-sidecar/`. Provides the production-tier Identity Sidecar binary (Section 10.6) with KMS / HSM / file key storage and the dual-proof post-quantum profile.

All three reference implementations interoperate at the credential wire format level. Test vectors at `test-vectors/` cover JCS canonicalization, the `eddsa-jcs-2022` cryptosuite, the dual-proof Ed25519 + ML-DSA-44 post-quantum profile (Section 13), and BitstringStatusList. A shared sidecar-contract test suite verifies that the Production-tier and Lightweight-tier sidecars (Section 10.6) accept and reject equivalent inputs.

These implementations are reference implementations only and do not imply endorsement of any specific deployment by their maintainers.

16.2 Developer Onboarding Tooling (Informative)

To lower the friction of evaluating and adopting this specification, four open-source developer-tooling packages accompany the protocol. Each package targets an AI tool ecosystem that developers already pay for and use daily, so the protocol's documentation, integration patterns, and audit guidance are available directly inside the developer's existing workflow without operating new infrastructure. The architectural pattern of distributing protocol-specific developer capabilities as packages consumed by the developer's own AI tool subscription, rather than as hosted services the protocol vendor operates, is documented in [PAD-057].

Surface	Where it runs	Source
Claude Skill	Anthropic Claude Code (CLI)	`https://github.com/vouch-protocol/vouch/tree/main/claude-skill`
OpenAI Custom GPT	ChatGPT (Plus / Team / Enterprise)	`https://github.com/vouch-protocol/vouch/tree/main/openai-gpt`
Gemini Gem	Google Gemini (Free / Advanced / Workspace)	`https://github.com/vouch-protocol/vouch/tree/main/gemini-gem`
Hosted Web Assistant	vouch-protocol.com (the protocol's website)	`https://github.com/vouch-protocol/vouch/tree/main/website-agent`

All four surfaces share a single canonical knowledge base (the eleven reference documents covering credential format, delegation, sidecar pattern, revocation, post-quantum profile, state verifiability, integrations, and troubleshooting), so the guidance is consistent regardless of which surface a developer chooses.

Capabilities exposed across these surfaces include: answering integration questions grounded in the protocol's documentation; generating starter code in Python, TypeScript, or Go; auditing a developer's repository for places where Vouch credentials

would be appropriate; explaining verification failures by mapping verifier reason codes to remediation steps; and (for the Hosted Web Assistant) signing real Vouch credentials as a live demonstration of the protocol.

For organizations evaluating the protocol for production deployment, these tooling surfaces materially reduce the proof-of-concept timeline. A developer can install the Claude Skill or build the Custom GPT in approximately ten minutes and receive protocol-grounded integration guidance specific to the organization's stack, without any data leaving the developer's existing AI subscription. This is intended to make implementer onboarding tractable for organizations that lack dedicated standards-implementation resources.

16.3 Implementer Interest, Sponsors, and Reviewers

Organizations indicating implementer interest, sponsorship of the work item, or formal review will be listed here as the work item progresses through CCG incubation. Organizations interested in being listed are invited to contact the editor at the address above.

---

17. Conformance Levels

A conforming Vouch Protocol implementation MUST satisfy one of three levels. Higher levels are strict supersets of lower levels: an L3 implementation satisfies all L2 and L1 requirements; an L2 satisfies all L1 requirements. Specifications referencing Vouch Protocol MAY require a specific minimum level; deployments SHOULD publish the level they target.

17.1 Level 1 (L1): Credential

The credential layer alone. Suitable for ad hoc agent-action attestation where the agent does not need continuous attestation and the deployment does not enforce structural capability bounds.

An L1-conforming implementation MUST:

- Issue and verify Vouch Credentials per Section 5 (Credential Format).
- Use the `eddsa-jcs-2022` cryptosuite (Ed25519 over RFC 8785 JCS-canonicalized payloads) per Section 7.
- Resolve issuer DIDs per Section 8.3, supporting at minimum `did:web` and `did:key`.
- Enforce credential expiry against `validFrom` / `validUntil` per Section 5.3.
- Enforce replay resistance through a recently-seen credential id cache per Section 12.2.

L1 implementations MAY omit: delegation chains, the Identity Sidecar pattern, status-list revocation, and the State Verifiability layer.

17.2 Level 2 (L2): Sidecar + Delegation + Revocation

The structural-security profile. Suitable for production deployments where the agent runtime includes an LLM and the deployment requires per-action capability bounds and revocation.

An L2-conforming implementation MUST satisfy all L1 requirements, plus:

- Deploy the Identity Sidecar pattern per Section 10: the signing key is held by a separate process; the LLM proposes intents over a local IPC channel; the Sidecar enforces a deployment-configured intent allow-list before signing.
- Support delegation chains per Section 9, including the resource-narrowing rule and the depth limit of five links.
- Support BitstringStatusList revocation per Section 11.2, including status-list polling on a configurable interval.
- Return structured rejection codes when refusing to sign (`intent_action_not_in_allowlist`, `intent_target_pattern_violation`, `intent_resource_out_of_scope`, `intent_missing_required_field`, `rate_limit_exceeded`, `delegation_chain_invalid`).

L2 implementations MAY omit: the dual-proof post-quantum profile, the Heartbeat Protocol, the validator quorum.

17.3 Level 3 (L3): State Verifiable + Post-Quantum

The full protocol. Suitable for long-running agents in regulated or adversarial environments, and for deployments preparing for the post-quantum migration.

An L3-conforming implementation MUST satisfy all L2 requirements, plus:

- Support the dual-proof post-quantum profile per Section 13: every Vouch Credential MAY carry two independent Data Integrity proofs on the same JCS-canonicalized bytes, one `eddsa-jcs-2022` and one `mldsa44-jcs-2026`. Verifiers MUST be able to operate in classical-only, PQ-only, or both-required modes (Section 13.2).
- Deploy the Heartbeat Protocol per Section 11.1: long-running agents renew their Session Voucher on a configurable interval (default 60 seconds).
- Enforce trust entropy decay per Section 11.5 against a deployment-configured threshold for each action class.
- Generate per-interval behavioural attestation digests and canary commit/reveal chains per Sections 11.3 and 11.7.
- Support M-of-N validator quorum per Section 11.6, where validators MAY specialize by role (policy, behaviour, budget).

L3 is the recommended target for any deployment processing high-stakes actions (financial transfers, regulated submissions, clinical record access, deploying production code).

17.4 Conformance Claim

A deployment claiming conformance at any level SHOULD publish a machine-readable declaration at a stable URL:

```
{
  "@context": ["https://vouch-protocol.com/contexts/conformance/v1"],
  "type": "VouchConformanceDeclaration",
  "deployment": "did:web:example.com",
  "level": "L2",
  "implementations": ["vouch-python==1.0.0", "go-sidecar==1.0.0"],
  "validated": "2026-05-25T00:00:00Z",
  "testVectorsPassing": [
    "jcs",
    "eddsa-jcs-2022",
    "bitstring-status-list",
    "delegation-chain",
    "sidecar-contract"
  ]
}
```

The deployment SHOULD re-validate against the cross-language test vector suite on each Vouch Protocol minor release.

---

18. Acknowledgements

The editor thanks the following individuals and organizations for review, feedback, and adjacent contributions to the agent identity ecosystem:

- Manu Sporny (Digital Bazaar), co-sponsor of this work item in the W3C Credentials Community Group, for review of the v1.6 draft and for foundational work on Verifiable Credentials, Data Integrity, and the `eddsa-jcs-2022` cryptosuite.
- The W3C Verifiable Credentials Working Group, for the body of standards on which this work composes.
- The W3C Credentials Community Group for hosting and reviewing this incubation effort.
- The IETF JOSE working group for ongoing work on PQ/T composite signatures.
- The C2PA technical committee for content provenance complementarity.
- *[Additional individuals to be listed as the document matures]*

Appendix A: Relationship to Existing Standards

This appendix describes how Vouch Protocol relates to existing standards. The framing is additive, Vouch Protocol is designed to compose with these specifications, not to replace them.

Standard	Relationship
W3C Verifiable Credentials Data Model 2.0	Vouch Credentials are W3C VCs. Full compliance with [VC-DATA-MODEL-2.0].
W3C Data Integrity	Vouch uses Data Integrity proofs ([VC-DATA-INTEGRITY]) with the `eddsa-jcs-2022` cryptosuite. No JOSE/JWS dependency.
W3C Decentralized Identifiers	Vouch uses DIDs as the identity format. Full DID Core compliance. `did:web` and `did:key` as primary methods.
W3C Controlled Identifiers (Multikey)	Vouch uses Multikey for verification methods. Algorithm-agnostic, supports Ed25519 and ML-DSA-44.
ZCAP-LD (Authorization Capabilities for Linked Data)	Vouch delegation chains share semantic intent with ZCAP-LD. Vouch uses JCS canonicalization rather than JSON-LD canonicalization, and requires explicit `resource` binding per link.
IETF JWS / JOSE (RFC 7515)	Vouch uses W3C Data Integrity proofs (`eddsa-jcs-2022`) rather than JWS Compact Serialization. Both are valid VC signature envelopes; Vouch chose Data Integrity for JSON-native canonicalization and cryptosuite agility.
IETF JCS (RFC 8785)	Vouch uses JCS canonicalization as required by `eddsa-jcs-2022`.
draft-ietf-jose-pq-composite-sigs	The PQ/T composite pattern in the IETF JOSE working group treats hybrid Ed25519 + ML-DSA-44 as a single composite signature; the Vouch dual-proof profile of Section 13 represents the same security guarantee as two independent W3C Data Integrity proofs over the same JCS-canonicalized bytes. The two designs are interchangeable in security level; Vouch chose dual proofs for alignment

Standard	Relationship
	with the W3C Data Integrity `proof`-as-array semantics.
W3C BitstringStatusList	Used for revocation via `credentialStatus`.
C2PA	Vouch provides identity for C2PA Content Credentials. Companion specification.
SPIFFE/SPIRE	Vouch extends workload identity concepts to AI agents with behavioral attestation.
OAuth 2.1	Vouch is complementary at the API layer. OAuth handles authorization grants for human-driven sessions; Vouch handles agent identity, intent attestation, and continuous state verifiability.

A.1 Out-of-Scope Companion Disclosures

A companion series of disclosures, [PAD-048] through [PAD-055], addresses operational governance of LLM coding assistants in developer workflows: in-session policy declaration via inline rule tags, write-only filesystem ledgers, deterministic egress interception at `git push` time, ephemeral rule semantics, hierarchical policy inheritance, and cross-session re-anchoring. These mechanisms are out of scope for this report (which addresses agent identity and accountability at runtime, not developer-time tooling), but are referenced for completeness. The full series and a reference implementation are available at `https://github.com/vouch-protocol/vouch/tree/main/docs/disclosures/`.

Where the coding-assistant series produces structured policy decisions that warrant cryptographic anchoring as long-term audit evidence (regulatory record-retention periods of 7+ years are common in regulated sectors), the **Vouch-Amnesia Attestation Bridge** described in [PAD-059] composes the coding-assistant series with the Vouch Credential format defined by this specification. The bridge signs the *decision output* of the deterministic policy evaluator (block / attest / allow) rather than the source events, binding it to a content-addressed hash of the active policy snapshot at decision time. When issued under the dual-proof post-quantum profile of Section 13, the resulting audit credentials remain verifiable against post-quantum adversaries without re-signing or re-issuance, making the Vouch Credential format a candidate substrate for multi-year audit log retention in regulated environments. The bridge itself is informative companion material; it does not impose any normative requirement on this specification, and conforming Vouch implementations are not required to support it.

The companion disclosure [PAD-060] (Single-Use Audited Override of a Deterministic Egress Block) addresses the operational pattern of one-time, time-bounded overrides of policy blocks and is bundled with the coding-assistant series for the same reason: it is operational tooling that composes with the Vouch Credential format but does not extend the protocol surface.

Appendix B: IANA Considerations

B.1 Media Type Registration

This specification registers the media type `application/vc+vouch` for transport of Vouch Credentials in HTTP request bodies. The identifier follows the `application/vc+<variant>` family pattern emerging across W3C Verifiable Credential subtypes (e.g. `application/vc+ld+json` for JSON-LD-canonicalized credentials, `application/vc+jwt` for JWT-encoded credentials), which a verifier can branch on with a single `Accept` / `Content-Type` parse.

The prior identifier `application/vouch+credential+json`, used by Vouch reference implementations through v1.6.x, is retained as a transitional alias for backward compatibility; new implementations SHOULD emit `application/vc+vouch` on the wire and accept either form during the transition window.

B.2 HTTP Header Field Registration

This specification registers the HTTP header field `Vouch-Credential` for header-based transport of Base64URL-encoded Vouch Credentials, where body transport is not possible.

B.3 Multicodec Registration

The multicodec entries used by this specification are:

- `mldsa44-pub`: registered (see the multicodec registry for the assigned code).
- `mldsa44-priv`: registered.

No composite hybrid multicodec is defined: the dual-proof profile of Section 13.2 uses two independent Multikey entries (one Ed25519, one ML-DSA-44) in the issuer's DID Document, each with its already-registered multicodec, so a separate `hybrid-...` codec is not needed.

B.4 W3C Data Integrity Cryptosuite Registration

This specification will coordinate with the W3C Data Integrity Working Group for the registration of: - `mldsa44-jcs-2026` (the post-quantum cryptosuite used as the second proof in the dual-proof profile of Section 13; to be aligned with the Digital Bazaar `mldsa44-rdfc-2024-cryptosuite` family's forthcoming JCS variant)

The transitional composite identifier `hybrid-eddsa-mldsa44-jcs-2026` is retained for v1.6.x reference-implementation backward compatibility only and will not be submitted for W3C registration.

Appendix C: Test Vectors

Test vectors are published in the companion repository at `https://github.com/vouch-protocol/vouch/tree/main/test-vectors/` and are REQUIRED for conformance testing. The test vector suite includes:

- **C.1 JCS canonicalization vectors**: Edge cases including Unicode normalization, numeric formatting, and key ordering, derived from RFC 8785 §3.2.
- **C.2 Ed25519 signing vectors**: Reference credentials with known keys and expected `proofValue` outputs.
- **C.3 DID Document examples**: `did:web` and `did:key` documents with Multikey verification methods.
- **C.4 Delegation chain vectors**: Linear chains of depth 1, 3, and 5 with valid and invalid resource-narrowing examples.
- **C.5 Heartbeat sequence vectors**: Sample heartbeat request/response pairs with canary reveal/commitment chains.
- **C.6 Dual-proof post-quantum vectors**: Reference credentials carrying two independent Data Integrity proofs (one `eddsa-jcs-2022`, one `mldsa44-jcs-2026`) over the same JCS-canonicalized bytes, published at `test-vectors/hybrid-eddsa-mldsa44/vector.json` with deterministic generation parameters and cross-implementation interop tests in Python, TypeScript, and Go.

Implementations claiming conformance MUST pass all test vectors in this revision. Cross-implementation interoperability testing is REQUIRED before an implementation may be listed in the Implementer Interest section.

References

Normative References

- **[BCP 14]** “Best Current Practice 14” aggregates the two keyword RFCs listed immediately below; the in-text citation `[BCP 14]` in Section 2 refers to this pair.
- **[RFC 2119]** Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997.
- **[RFC 8174]** Leiba, B., “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words”, BCP 14, RFC 8174, May 2017.
- **[RFC 8032]** Josefsson, S. and I. Liusvaara, “Edwards-Curve Digital Signature Algorithm (EdDSA)”, RFC 8032, January 2017.
- **[RFC 8785]** Rundgren, A., Jordan, B., and S. Erdtman, “JSON Canonicalization Scheme (JCS)”, RFC 8785, June 2020.

- **[VC-DATA-MODEL-2.0]** Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, “Verifiable Credentials Data Model v2.0”, W3C Recommendation, 2024.
- **[VC-DATA-INTEGRITY]** Longley, D. and M. Sporny, “Verifiable Credential Data Integrity 1.0”, W3C Recommendation.
- **[VC-DI-EDDSA]** Longley, D. and M. Sporny, “Data Integrity EdDSA Cryptosuites v1.0”, W3C Recommendation. https://www.w3.org/TR/vc-di-eddsa/
- **[W3C DID Core]** Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, “Decentralized Identifiers (DIDs) v1.0”, W3C Recommendation, 19 July 2022.
- **[W3C Controlled Identifiers]** “Controlled Identifiers (Multikey, JsonWebKey)”, W3C Working Draft.
- **[FIPS 204]** National Institute of Standards and Technology, “Module-Lattice-Based Digital Signature Standard”, FIPS 204, August 2024.

Informative References

- **[DID Web]** Caballero, O., “did:web Method Specification”, W3C Community Group Report.
- **[ZCAP-LD]** Longley, D. and M. Sporny, “Authorization Capabilities for Linked Data”, W3C Community Group Report.
- **[VC-BITSTRING-STATUS-LIST]** “Bitstring Status List v1.0”, W3C Recommendation.
- **[draft-ietf-jose-pq-composite-sigs]** IETF JOSE Working Group, “PQ/T Composite Signatures for JOSE”, Internet-Draft.
- **[C2PA]** “C2PA Technical Specification”, Coalition for Content Provenance and Authenticity, v2.1, 2024.
- **[MCP]** “Model Context Protocol”, Anthropic, 2024.

Defensive Disclosures (Vouch Protocol Prior Art Portfolio)

The following defensive disclosures, published under CC0 to the Vouch Protocol prior art portfolio, are referenced as informative context for specific architectural choices. The complete portfolio of 60 disclosures is available at https://github.com/vouch-protocol/vouch/tree/main/docs/disclosures/. Disclosures cited inline above are listed individually below; others in the portfolio are companion material outside the scope of this report.

- **[PAD-003]** Identity Sidecar Pattern.
- **[PAD-010]** Semantic Consent Signing.
- **[PAD-016]** Dynamic Credential Renewal (Heartbeat Protocol).
- **[PAD-020]** Ratchet Lock Protocol.
- **[PAD-021]** Inverse Capability Protocol.
- **[PAD-022]** Swarm Limits Protocol.
- **[PAD-030]** Zero-Knowledge Reputation Portability.

- **[PAD-032]** Cryptographic Mortality Protocol.
- **[PAD-033]** ZK Post-Quantum Signature Compression.
- **[PAD-034]** Composite Threshold Swarm Consensus.
- **[PAD-035]** Asynchronous Chunked Edge Post-Quantum Signatures.
- **[PAD-036]** Aggregated Reputation Scoring via Verifiable State Receipts.
- **[PAD-037]** Cross-Protocol Agent Credential Federation.
- **[PAD-038]** Decentralized Agent Capability Discovery Protocol.
- **[PAD-039]** JCS-Deterministic Multi-Party Trust State.
- **[PAD-040]** Dual-Proof Same-Canonical-Bytes Property (Ed25519 + ML-DSA-44 over JCS; formerly framed as “hybrid composite”).
- **[PAD-041]** Algorithm-Agnostic Verification via Multikey Multicodec Discrimination.
- **[PAD-042]** Standardized Metadata Schema for AI Agent Ledger Signatures.
- **[PAD-043]** Cryptographic Weight Binding for Model-Intrinsic AI Identity.
- **[PAD-044]** Ephemeral ZK-State Channels for Agentic Layer 2 Scalability.
- **[PAD-045]** Proof of Non-Hallucination via Cryptographic Retrieval Anchoring.
- **[PAD-046]** Algorithm Quorum Verification via M-of-N Cryptosuite Diversity.
- **[PAD-047]** Verifiable Delay Functions for Cryptographic Rate-Limiting of Autonomous Agent Actions.

Companion Series: LLM Coding-Assistant Governance (Out of Scope for this Report)

The following disclosures form a companion series addressing operational governance of LLM coding assistants in developer workflows. They are referenced in Appendix A.1 as out-of-scope companion material; they do not change any normative requirement of this specification.

- **[PAD-048]** Write-Only Asynchronous Context Ledger for LLM Coding Assistants.
- **[PAD-049]** Decoupled Semantic Policy Extraction via Passive Source Monitoring.
- **[PAD-050]** Zero-Context Deterministic Egress Interception via Pre-Push Hook.
- **[PAD-051]** Parallel Intent Extraction via a Local Shadow Small Language Model.
- **[PAD-052]** UI State Sniffing for Policy Extraction from Closed-Box AI Coding Applications.
- **[PAD-053]** Time-Bounded Ephemeral Rules with Auto-Expiry for LLM Coding Assistant Sessions.
- **[PAD-054]** Filesystem-Hierarchy Policy Inheritance for LLM Coding Assistant Workspaces.
- **[PAD-055]** Cross-Session Policy Re-Anchoring via Pre-Flight Context Replay.
- **[PAD-059]** Vouch-Amnesia Attestation Bridge for Deterministic Pre-Push Policy Decisions.
- **[PAD-060]** Single-Use Audited Override of a Deterministic Egress Block.

Companion Series: AI Assistant Architecture and Identity Hygiene (Out of Scope for this Report)

The following disclosures address operational deployment of Vouch-aware AI assistants and key-leak handling. They build on this specification's primitives but address operational concerns outside the protocol surface.

- **[PAD-056]** Capability-Bounded AI Assistant Output via Intent Allow-List at the Identity Sidecar.
- **[PAD-057]** Bring-Your-Own-LLM Distribution of Protocol Capabilities via AI Tool Packaging.
- **[PAD-058]** Automated DID Rotation and Verifier Broadcast Pipeline on Static Leak Detection.
