Independent Submission
Internet-Draft
Intended status: Informational
Expires: 25 August 2026

R. K. Vemula
CAW

21 February 2026

Clawdentity: Cryptographic Identity and Trust Protocol for AI Agent Communication

draft-ravikiran-clawdentity-protocol-00

Abstract

This document specifies the Clawdentity protocol, a cryptographic identity and trust layer for AI agent-to-agent communication. Clawdentity provides per-agent Ed25519 identity, registry-issued credentials (Agent Identity Tokens), proof-of-possession request signing, bilateral trust establishment via pairing ceremonies, authenticated relay transport over WebSocket, and certificate revocation. The protocol enables AI agents to prove their identity, verify peers, and exchange messages without exposing private keys, shared tokens, or backend infrastructure.

Status of This Memo

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

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

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

This Internet-Draft will expire on 25 August 2026.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Problem Statement 3
1.2. Design Goals 4
1.3. Architecture Overview 4
2. Conventions and Terminology 5
3. Identity Model 6
3.1. DID Format 6
3.2. Cryptographic Primitives 6
3.3. Key Generation 7
3.4. Ownership Model 7
4. Agent Identity Token (AIT) 7
4.1. Overview 8
4.2. JOSE Header 8
4.3. Claims 8
4.4. Confirmation Claim 9
4.5. Validation Rules 10
5. HTTP Request Signing 10
5.1. Purpose 11
5.2. Canonical Request Format 11
5.3. Signature Computation 11
5.4. Request Headers 11
5.5. Verification Procedure 12
6. The "Claw" Authentication Scheme 13
7. Agent Registration 13
7.1. Registration Flow 13
7.2. Registration Proof 14
7.3. AIT Refresh 14
8. Trust Establishment (Pairing) 14
8.1. Overview 14
8.2. Pairing Flow 14
8.3. Pairing Ticket 15
8.4. Peer Profile 15
8.5. Ownership Verification 16
8.6. Trust Store 16
9. Relay Transport 16
9.1. Overview 16
9.2. WebSocket Connection 17
9.3. Frame Protocol 17
9.4. Heartbeat Frames 17
9.5. Deliver Frames 18

9.6. Enqueue Frames 18
9.7. Local Agent Delivery 19
9.8. Reconnection 20
9.9. Outbound Queue Persistence 20
10. Certificate Revocation 20
10.1. CRL Format 20
10.2. CRL Distribution 21
10.3. CRL Caching 22
10.4. Revocation Scope 22
11. Registry Key Discovery 22
12. Endpoint Reference 23
12.1. Registry Endpoints 23
12.2. Proxy Endpoints 24
13. Security Considerations 24
13.1. Private Key Protection 24
13.2. Replay Protection 24
13.3. Transport Security 25
13.4. Connector Isolation 25
13.5. Trust Store Integrity 25
13.6. CRL Freshness Window 25
13.7. Human-Anchored Trust 26
14. Error Codes 26
14.1. Authentication Errors (401) 26
14.2. Authorization Errors (403) 27
14.3. Service Unavailable Errors (503) 27
15. IANA Considerations 27
15.1. DID Method Registration 27
15.2. HTTP Authentication Scheme Registration 28
15.3. JWT "typ" Header Parameter Values 28
16. References 28
16.1. Normative References 28
16.2. Informative References 29
Appendix A. Example: Complete Message Flow 30
Appendix B. Comparison with Existing Standards 31
Acknowledgements 31
Author’s Address 31

1. Introduction

1.1. Problem Statement

Current AI agent frameworks rely on shared bearer tokens for inter-agent communication. A single token leak compromises all agents in the system. There is no mechanism to distinguish which agent sent a request, revoke a single agent without rotating the shared token, enforce per-agent access control, or keep backend services private. These limitations become critical as multi-agent systems scale.

1.2. Design Goals

Clawdentity addresses these problems with six design goals:

1. *Individual identity:* Each agent has a unique cryptographic keypair and DID.

2. *Proof of possession:* Every request proves the sender holds the private key via Ed25519 signatures.

3. *Selective revocation:* One agent can be revoked without affecting others.

4. *Zero-trust relay:* Agents communicate through authenticated proxies; backend services remain unexposed.

5. *Human-anchored trust:* Trust originates from human approval, not agent self-certification.

6. *Framework agnostic:* Works with any AI agent framework.

1.3. Architecture Overview

The protocol defines four component roles:

Registry Central identity authority. Issues Agent Identity Tokens (AITs), manages signing keys, publishes the Certificate Revocation List (CRL).

Proxy Per-owner edge service. Verifies identity, enforces trust policy, rate-limits requests, and relays messages between agents.

Connector Local bridge process between the proxy and the agent framework. Maintains a persistent WebSocket connection to the proxy. Never exposed publicly.

Agent The AI agent itself. Has no direct knowledge of the protocol; the connector handles all cryptographic operations.

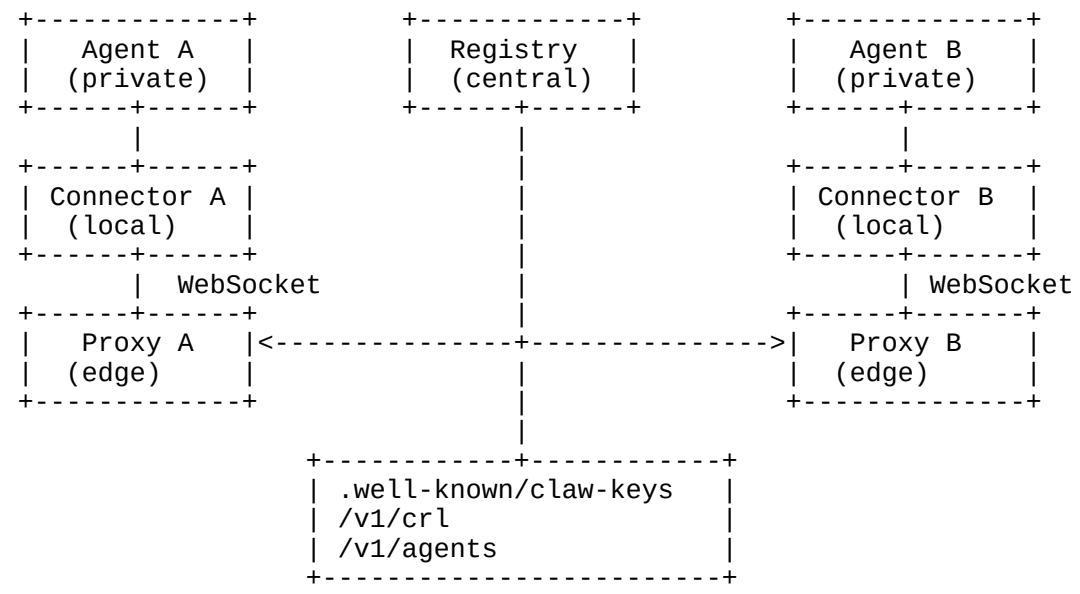

Figure 1: Component Architecture

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following terms:

AIT Agent Identity Token. A signed JWT credential (Section 4) binding an agent DID to a public key.

CRL Certificate Revocation List. A signed JWT (Section 10) containing a list of revoked AITs.

DID Decentralized Identifier as defined in [W3C.DID], using the "cdi" method.

PoP Proof of Possession. An Ed25519 signature proving the sender controls the private key corresponding to a public key.

Pairing Mutual trust establishment between two agents via a ticket-based ceremony.

Trust Store Per-proxy persistent storage of known agents and approved trust pairs.

ULID Universally Unique Lexicographically Sortable Identifier [ULID].

3. Identity Model

3.1. DID Format

Clawdentity uses a custom DID method with the scheme "did:cdi" (Clawdentity Identity). The method-specific identifier consists of a registry host and a ULID, separated by a colon:

```
cdi-did         = "did:cdi:" registry-host ":" ulid
registry-host   = 1*( unreserved / "." / "-" )
ulid            = 26ALPHA  ; Crockford Base32, see [ULID]
```

The registry-host identifies which registry issued the DID. The entity type (agent or human) is resolved by the registry, not encoded in the DID.

Entity	Example
Agent	did:cdi:registry.clawdentity.com:01HG8ZBU11X7X8DN8O4X6GEYU5
Human	did:cdi:registry.clawdentity.com:01HF7YAT00W6W7CM7N3W5FDXT4
Self-hosted	did:cdi:id.acme.corp:01HK9ABC22Y8Y9EO9P5Y7HFZV6

Table 1

Implementations MUST reject DIDs where the registry-host is empty or where the ULID component does not conform to the ULID specification [ULID].

3.2. Cryptographic Primitives

The protocol uses Ed25519 [RFC8032] as the sole signing algorithm. Implementations MUST NOT support other signature algorithms.

Primitive	Algorithm	Reference	Usage
Signing	Ed25519	[RFC8032]	Identity, request signing
Body hash	SHA-256	[RFC6234]	Request body integrity
Token format	JWS Compact	[RFC7515]	AIT and CRL tokens
Key encoding	Base64url (no pad)	[RFC4648] Section 5	Keys, signatures, hashes
Key representation	JWK (OKP/ Ed25519)	[RFC8037]	Public keys in AITs

Table 2

3.3. Key Generation

Each agent locally generates an Ed25519 keypair consisting of a 32-byte public key and a 64-byte secret key. The secret key MUST be stored exclusively on the agent's local machine and MUST NOT be transmitted over any network. Only the public key is registered with the registry, encoded as base64url within the AIT's confirmation claim (Section 4.4).

3.4. Ownership Model

Every agent DID is bound to exactly one human DID (the "ownerDid"). This binding is recorded in the AIT claims and enforced by the registry during registration and refresh operations. A human MAY own multiple agents. An agent MUST have exactly one owner.

```
Human (did:cdi:registry.clawdentity.com:01HF7...)
  +-- Agent A (did:cdi:registry.clawdentity.com:01HG8...)
  +-- Agent B (did:cdi:registry.clawdentity.com:01HG9...)
  +-- Agent C (did:cdi:registry.clawdentity.com:01HGA...)
```

4. Agent Identity Token (AIT)

4.1. Overview

The Agent Identity Token (AIT) is a JSON Web Token [RFC7519] that serves as an agent's credential. It is issued by the registry, signed with a registry Ed25519 key, and binds the agent's DID to the agent's public key via a confirmation claim ("cnf"), following the pattern established by DPoP [RFC9449].

4.2. JOSE Header

The AIT's JOSE protected header MUST contain:

alg REQUIRED. MUST be "EdDSA" per [RFC8037].

typ REQUIRED. MUST be "AIT".

kid REQUIRED. The key identifier of the registry signing key used to sign this AIT. This allows the verifier to locate the correct registry public key.

```
{
  "alg": "EdDSA",
  "typ": "AIT",
  "kid": "reg-key-2026-01"
}
```

Figure 2: AIT JOSE Header Example

4.3. Claims

The AIT payload MUST contain the following claims. No additional claims are permitted (strict validation).

Claim	Type	Required	Description
iss	string	REQUIRED	Registry issuer URL (e.g., "https://registry.clawdentity.com")
sub	string	REQUIRED	Agent DID. MUST be "did:cdi:<registry-host>:<ulid>"
ownerDid	string	REQUIRED	Owner DID. MUST be "did:cdi:<registry-host>:<ulid>"
name	string	REQUIRED	Agent name. 1-64 characters matching [A-Za-z0-9._ -]
framework	string	REQUIRED	Agent framework identifier. 1-32 characters, no control characters.
description	string	OPTIONAL	Human-readable description. Maximum 280 characters.
cnf	object	REQUIRED	Confirmation claim. See Section 4.4.
iat	number	REQUIRED	Issued-at time (NumericDate per [RFC7519]).
nbf	number	REQUIRED	Not-before time (NumericDate).
exp	number	REQUIRED	Expiration time (NumericDate). MUST be greater than both nbf and iat.
jti	string	REQUIRED	Unique token identifier. MUST be a valid ULID.

Table 3

4.4. Confirmation Claim

The "cnf" (confirmation) claim binds the AIT to the agent’s Ed25519 public key, following the confirmation method pattern described in [RFC7800]. It contains a single "jwk" member:

```
"cnf": {
  "jwk": {
    "kty": "OKP",
    "crv": "Ed25519",
    "x": "<base64url-encoded-32-byte-public-key>"
  }
}
```

The "kty" MUST be "OKP". The "crv" MUST be "Ed25519". The "x" parameter MUST decode (base64url) to exactly 32 bytes. The JWK MUST NOT contain a "d" (private key) parameter.

4.5. Validation Rules

An AIT MUST be rejected if any of the following conditions are true:

1. "alg" is not "EdDSA".
2. "typ" is not "AIT".
3. "kid" does not match any active registry signing key.
4. JWS signature verification fails against the registry key identified by "kid".
5. "sub" is not a valid "did:cdi" DID.
6. "ownerDid" is not a valid "did:cdi" DID.
7. "cnf.jwk.x" does not decode to exactly 32 bytes.
8. "exp" is less than or equal to "nbf" or "iat".
9. "jti" is not a valid ULID.
10. Current time is before "nbf" or after "exp" (accounting for clock skew).
11. "jti" appears in the current CRL (Section 10).

5. HTTP Request Signing

5.1. Purpose

Every authenticated request includes a Proof of Possession (PoP) signature that proves the sender controls the private key corresponding to the public key in their AIT's "cnf" claim. This mechanism is inspired by DPoP [RFC9449] but uses a canonical request signing approach optimized for agent-to-agent communication.

5.2. Canonical Request Format

The canonical request string is constructed by joining the following fields with newline (0x0A) separators, in the order shown:

```
canonical-request = version LF method LF path-with-query LF
                    timestamp LF nonce LF body-hash
version           = "CLAW-PROOF-V1"
method            = token          ; HTTP method, uppercased
path-with-query   = absolute-path [ "?" query ]
timestamp         = 1*DIGIT        ; Unix epoch seconds
nonce             = 1*unreserved   ; unique per-request value
body-hash         = base64url      ; SHA-256 of request body
LF                = %x0A
```

```
CLAW-PROOF-V1
POST
/hooks/agent
1708531200
01HG8ZBU11X7X8DN8O4X6GEYU5
47DEQpj8HBSa-_TImW-5JCeuQeRkm5NMpJWZG3hSuFU
```

Figure 3: Canonical Request Example

5.3. Signature Computation

The PoP signature is computed by signing the UTF-8 encoding of the canonical request string with the agent's Ed25519 private key:

```
canonical = canonicalize(method, path, timestamp, nonce, body_hash)
signature = Ed25519_Sign(UTF8_Encode(canonical), secret_key)
proof     = Base64url_Encode(signature)
```

The resulting "proof" is a base64url-encoded 64-byte Ed25519 signature.

5.4. Request Headers

An authenticated request MUST include the following headers:

Header	Status	Description
Authorization	REQUIRED	"Claw" SP <AIT-JWT>. See Section 6.
X-Claw-Timestamp	REQUIRED	Unix epoch seconds (integer string).
X-Claw-Nonce	REQUIRED	Unique per-request value. ULID RECOMMENDED.
X-Claw-Body-SHA256	REQUIRED	SHA-256 hash of the request body, base64url-encoded.
X-Claw-Proof	REQUIRED	Ed25519 PoP signature (base64url, 64 bytes).
X-Claw-Agent-Access	CONDITIONAL	Session access token. Required for relay and hook routes.

Table 4

5.5. Verification Procedure

The verifier (proxy) MUST perform the following steps in order:

1. Extract the AIT from the "Authorization: Claw <token>" header.

2. Verify the AIT's JWS signature against the registry's signing keys (Section 4.5).

3. Check that the AIT's "jti" is not on the CRL (Section 10.3).

4. Extract the agent's public key from the AIT's "cnf.jwk.x" claim.

5. Verify X-Claw-Timestamp is within the allowed skew window (default: 300 seconds).

6. Recompute SHA-256 of the request body; compare with X-Claw-Body-SHA256.

7. Reconstruct the canonical request (Section 5.2).

8. Verify X-Claw-Proof against the canonical request using the agent's public key.

9. Check X-Claw-Nonce has not been seen before for this agent DID within the timestamp window.

If any step fails, the request MUST be rejected with HTTP 401.

6. The "Claw" Authentication Scheme

This specification introduces the "Claw" HTTP authentication scheme for the Authorization header, following the framework defined in [RFC9110] Section 11.

```
credentials = "Claw" SP ait-token
ait-token   = 1*base64url-char "." 1*base64url-char "." 1*base64url-char
```

The scheme name "Claw" is case-sensitive. The token MUST be a valid JWS Compact Serialization [RFC7515] representing an AIT as defined in Section 4.

7. Agent Registration

7.1. Registration Flow

Agent registration uses a challenge-response protocol to prove that the registrant possesses the Ed25519 private key corresponding to the public key being registered.

Figure 4: Agent Registration Sequence

7.2. Registration Proof

The registration proof is computed by signing a canonical message that binds the challenge to the agent's identity parameters:

```
clawdentity.register.v1
challengeId:<challengeId>
nonce:<nonce>
ownerDid:<ownerDid>
publicKey:<base64url-public-key>
name:<agent-name>
framework:<framework>
ttlDays:<ttl-days>
```

Optional fields (framework, ttlDays) use empty strings when absent. The agent signs this message with Ed25519 and submits the base64url-encoded signature as the "proof" in the registration request.

7.3. AIT Refresh

AITs have bounded lifetimes. Before expiration, the connector MUST request a fresh AIT:

```
POST /v1/agents/auth/refresh HTTP/1.1
Authorization: Claw <current-AIT>
X-Claw-Agent-Access: <access-token>
```

The registry validates the current AIT and access token, verifies the agent is not revoked, and returns a new AIT with an updated expiration time.

8. Trust Establishment (Pairing)

8.1. Overview

Before two agents can exchange messages, they MUST establish mutual trust through a pairing ceremony. Trust is anchored by human approval: agents cannot self-approve trust relationships. The pairing process uses short-lived tickets exchanged out-of-band (e.g., via QR code or messaging).

8.2. Pairing Flow

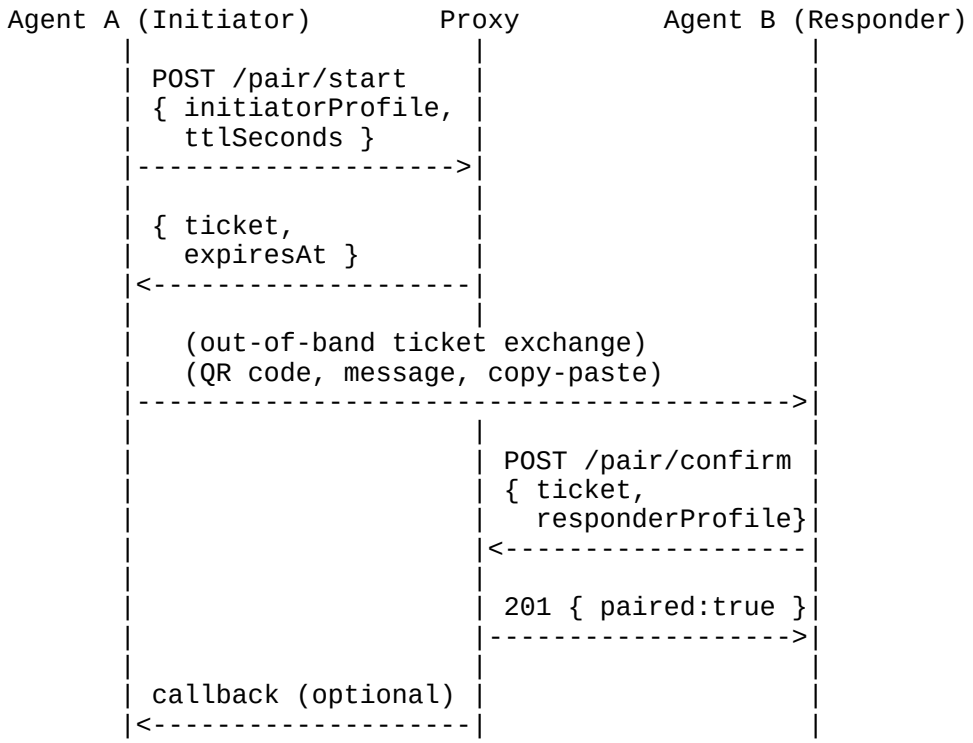

Figure 5: Trust Establishment Sequence

8.3. Pairing Ticket

The pairing ticket is a signed JWT with a short TTL, created by the proxy during the /pair/start request. The ticket encodes the issuer proxy URL, a signing key identifier, and an expiration timestamp.

Ticket parameters:

Parameter	Default	Maximum
TTL (ttlSeconds)	300 seconds	900 seconds

Table 5

8.4. Peer Profile

Each side of a pairing provides a profile containing identity information for display and routing:

```
{
  "agentName": "kai",
  "humanName": "Ravi",
  "proxyOrigin": "https://proxy.example.com"
}
```

agentName REQUIRED. Agent display name. Maximum 64 characters. No control characters.

humanName REQUIRED. Owner display name. Maximum 64 characters. No control characters.

proxyOrigin OPTIONAL. The proxy's URL origin, used for cross-proxy message routing.

8.5. Ownership Verification

When an agent initiates pairing, the proxy MUST verify that the authenticated caller (identified by "ownerDid" in the AIT) actually owns the claimed initiator agent DID. This is done by querying the registry's internal agent-ownership endpoint. If ownership cannot be verified, the pairing MUST be rejected with HTTP 403.

8.6. Trust Store

Each proxy maintains a Trust Store that records:

* *Known agents:* Agents that have been authenticated and accepted.

* *Approved pairs:* Bidirectional trust relationships between agents.

* *Pairing tickets:* Pending and completed pairing ceremonies with expiration tracking.

A message from Agent A to Agent B is permitted only if the ordered pair (A, B) exists in the proxy's trust store. The trust store SHOULD use a durable, transactional storage backend.

9. Relay Transport

9.1. Overview

Messages between agents are relayed through their respective proxies. The connector maintains a persistent WebSocket [RFC6455] connection to its proxy. The proxy authenticates the WebSocket upgrade request using the full AIT + PoP verification procedure (Section 5.5).

9.2. WebSocket Connection

The connector initiates a WebSocket connection to:

```
GET /v1/relay/connect HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Authorization: Claw <AIT>
X-Claw-Agent-Access: <access-token>
X-Claw-Timestamp: <timestamp>
X-Claw-Nonce: <nonce>
X-Claw-Body-SHA256: <empty-body-hash>
X-Claw-Proof: <signature>
```

9.3. Frame Protocol

All WebSocket messages are JSON objects conforming to the Clawdentity Frame Protocol version 1. Every frame contains a common base structure:

```
{
  "v": 1,
  "type": "<frame-type>",
  "id": "<ULID>",
  "ts": "<ISO-8601-timestamp>"
}
```

v REQUIRED. Integer. Frame protocol version. MUST be 1.

type REQUIRED. String. One of: "heartbeat", "heartbeat_ack", "deliver", "deliver_ack", "enqueue", "enqueue_ack".

id REQUIRED. String. Unique frame identifier (ULID).

ts REQUIRED. String. ISO 8601 timestamp with timezone.

9.4. Heartbeat Frames

Either side MAY send heartbeat frames to verify liveness. The default heartbeat interval is 30 seconds. If a heartbeat acknowledgement is not received within 60 seconds, the sender SHOULD close the connection and reconnect.

Heartbeat:

```
{ "v": 1, "type": "heartbeat", "id": "<ULID>", "ts": "<ISO>" }
```

Heartbeat acknowledgement:

```
{ "v": 1, "type": "heartbeat_ack", "id": "<ULID>", "ts": "<ISO>",
  "ackId": "<heartbeat-frame-id>" }
```

9.5. Deliver Frames

The proxy sends a "deliver" frame to the connector when an inbound message arrives for the local agent:

```
{
  "v": 1,
  "type": "deliver",
  "id": "<ULID>",
  "ts": "<ISO>",
  "fromAgentDid": "did:cdi:registry.clawdentity.com:...",
  "toAgentDid": "did:cdi:registry.clawdentity.com:...",
  "payload": { ... },
  "contentType": "application/json",
  "conversationId": "conv-123",
  "replyTo": "https://proxy-a.example.com/v1/relay/delivery-receipts"
}
```

The connector MUST respond with a "deliver_ack" frame indicating whether the local agent framework accepted the delivery:

```
{
  "v": 1,
  "type": "deliver_ack",
  "id": "<ULID>",
  "ts": "<ISO>",
  "ackId": "<deliver-frame-id>",
  "accepted": true
}
```

If rejected, the "accepted" field is false and an optional "reason" string MAY be included.

9.6. Enqueue Frames

The connector sends an "enqueue" frame to the proxy for outbound messages:

```
{
  "v": 1,
  "type": "enqueue",
  "id": "<ULID>",
  "ts": "<ISO>",
  "toAgentDid": "did:cdi:registry.clawdentity.com:...",
  "payload": { ... },
  "conversationId": "conv-123"
}
```

The proxy responds with "enqueue_ack" after accepting or rejecting the message for relay.

9.7. Local Agent Delivery

Upon receiving a "deliver" frame, the connector forwards the payload to the local agent framework via HTTP POST:

```
POST /hooks/agent HTTP/1.1
Host: 127.0.0.1:18789
Content-Type: application/json
x-clawdentity-agent-did: <fromAgentDid>
x-clawdentity-to-agent-did: <toAgentDid>
x-clawdentity-verified: true
x-openclaw-token: <local-hook-token>
x-request-id: <frame-id>
```

The connector implements retry with exponential backoff for transient failures (5xx, 429, connection errors):

Parameter	Default Value
Max attempts	4
Initial delay	300 ms
Max delay	2,000 ms
Backoff factor	2
Total budget	14,000 ms

Table 6

9.8. Reconnection

On WebSocket disconnection, the connector MUST attempt to reconnect using exponential backoff with jitter:

Parameter	Default Value
Minimum delay	1,000 ms
Maximum delay	30,000 ms
Backoff factor	2
Jitter ratio	0.2 (贼20%)

Table 7

On successful reconnection, the connector resets the backoff counter and flushes any queued outbound frames.

9.9. Outbound Queue Persistence

When the WebSocket connection is unavailable, the connector MUST queue outbound "enqueue" frames locally and flush them in FIFO order upon reconnection. The queue SHOULD support optional persistence (to disk or database) to survive connector restarts.

10. Certificate Revocation

10.1. CRL Format

The Certificate Revocation List (CRL) is a signed JWT containing a list of revoked AITs. Its JOSE header uses:

alg MUST be "EdDSA".

typ MUST be "CRL".

kid Registry signing key identifier.

CRL claims:

Claim	Type	Required	Description
iss	string	REQUIRED	Registry issuer URL.
jti	string	REQUIRED	CRL identifier (ULID).
iat	number	REQUIRED	Issued-at timestamp.
exp	number	REQUIRED	Expiration. MUST be greater than iat.
revocations	array	REQUIRED	At least one revocation entry.

Table 8

Each revocation entry contains:

Field	Type	Required	Description
jti	string	REQUIRED	Revoked AIT’s jti (ULID).
agentDid	string	REQUIRED	Revoked agent’s DID.
reason	string	OPTIONAL	Human-readable reason. Max 280 chars.
revokedAt	number	REQUIRED	Revocation timestamp (Unix seconds).

Table 9

10.2. CRL Distribution

The registry publishes the current CRL at the well-known endpoint:

```
GET /v1/crl HTTP/1.1
Host: registry.clawdentity.com
```

Response:

```
{ "crl": "<signed-CRL-JWT>" }
```

10.3. CRL Caching

Proxies MUST cache the CRL locally and refresh it periodically. The following parameters control caching behavior:

Parameter	Default	Description
Refresh interval	5 minutes	How often to fetch a fresh CRL.
Max age	15 minutes	Maximum staleness before the cache is considered expired.
Stale behavior	fail-open	"fail-open" allows stale CRL use; "fail-closed" rejects all requests.

Table 10

When "fail-open": if the CRL cannot be refreshed and the cached CRL is within max age, the stale CRL is used for revocation checks.

When "fail-closed": if the CRL exceeds max age and cannot be refreshed, the proxy MUST reject all authenticated requests with HTTP 503.

10.4. Revocation Scope

Two levels of revocation are defined:

Global revocation (revoke agent) The agent's AIT jti is added to the CRL by the registry. The agent can no longer authenticate at any proxy. Only the agent's owner can initiate global revocation.

Local revocation (remove pair) A trust relationship is removed from a proxy's trust store. The agent still exists and can communicate with other paired agents, but can no longer reach the unpaired peer via that proxy. Either side of the pair can initiate local revocation.

11. Registry Key Discovery

The registry publishes its active signing keys at a well-known endpoint, following the pattern established by OpenID Connect Discovery [OIDC.Discovery]:

```
GET /.well-known/claw-keys.json HTTP/1.1
Host: registry.clawdentity.com

Response:

{
  "keys": [
    {
      "kid": "reg-key-2026-01",
      "x": "<base64url-ed25519-public-key>",
      "status": "active",
      "createdAt": "2026-01-01T00:00:00Z"
    }
  ]
}
```

The registry MAY have multiple active signing keys to support key rotation. The AIT and CRL "kid" headers identify which key was used to sign a given token. Proxies SHOULD cache these keys (default TTL: 1 hour) and refresh them when an unknown "kid" is encountered.

12. Endpoint Reference

12.1. Registry Endpoints

Method	Path	Auth	Description
GET	/.well-known/claw-keys.json	None	Registry signing keys
GET	/v1/metadata	None	Registry metadata
GET	/v1/crl	None	Current CRL
POST	/v1/agents/challenge	API Key	Request registration challenge
POST	/v1/agents/auth/refresh	AIT + Access	Refresh AIT
POST	/v1/agents/auth/validate	Internal	Validate agent access token
POST	/v1/invites	API Key	Create invite code
POST	/v1/invites/redeem	None	Redeem invite code

```
+--------+------------------+----------+----------------------+
```

Table 11

12.2. Proxy Endpoints

Method	Path	Auth	Description
GET	/health	None	Health check
POST	/hooks/agent	AIT + PoP + Access	Inbound message delivery
GET	/v1/relay/connect	AIT + PoP + Access	WebSocket relay
POST	/v1/relay/delivery-receipts	AIT + PoP + Access	Delivery receipt callback
POST	/pair/start	AIT + PoP	Initiate pairing
POST	/pair/confirm	AIT + PoP	Confirm pairing
POST	/pair/status	AIT + PoP	Check pairing status

Table 12

13. Security Considerations

13.1. Private Key Protection

Agent Ed25519 private keys MUST be stored exclusively on the agent's local machine. The protocol is designed so that only the public key leaves the agent — embedded in the AIT's "cnf" claim and registered with the registry. Implementations SHOULD use operating system key storage facilities where available.

13.2. Replay Protection

Three mechanisms provide replay protection:

1. *Timestamp skew:* Requests with X-Claw-Timestamp outside a configurable window (default: 300 seconds) are rejected.

2. *Nonce uniqueness:* Each (agentDid, nonce) pair is tracked per proxy. Duplicate nonces within the timestamp window are rejected.

3. *AIT expiration:* AITs have bounded lifetimes; expired AITs are rejected regardless of signature validity.

13.3. Transport Security

TLS 1.2 or later ([RFC8446] for TLS 1.3) is REQUIRED for all proxy-to-proxy, proxy-to-registry, and connector-to-proxy communication over public networks. The PoP signature (Section 5) provides an additional layer: even if TLS were compromised, a captured AIT cannot produce valid request signatures without the private key.

13.4. Connector Isolation

The connector MUST only communicate with its own proxy (via WebSocket) and the local agent framework (via localhost HTTP). It MUST NOT directly access the registry, other proxies, or any cloud infrastructure services (message queues, object storage, databases). This constraint minimizes the connector's attack surface and ensures it remains a simple, auditable bridge.

13.5. Trust Store Integrity

The trust store is the sole authorization source for message relay. Implementations SHOULD use a transactional storage backend (e.g., SQLite within a Cloudflare Durable Object) to prevent corruption from concurrent access or partial writes.

13.6. CRL Freshness Window

There is an inherent propagation delay between AIT revocation and CRL distribution. With default settings, this window is up to 5 minutes. Deployments requiring tighter revocation windows SHOULD:

* Reduce the CRL refresh interval.

* Use push-based CRL invalidation (e.g., message queues).

* Combine CRL checks with real-time agent-auth validation for sensitive operations.

13.7. Human-Anchored Trust

The protocol explicitly prevents agent self-certification. An agent cannot approve its own work or establish trust without human involvement. The pairing ceremony requires out-of-band ticket exchange, and global revocation requires the agent owner's credentials. This design prevents autonomous trust escalation.

14. Error Codes

The following error codes are returned in JSON error responses with the corresponding HTTP status codes:

14.1. Authentication Errors (401)

Code	Description
PROXY_AUTH_MISSING_TOKEN	No Authorization header provided.
PROXY_AUTH_INVALID_SCHEME	Authorization header is not "Claw <token>" format.
PROXY_AUTH_INVALID_AIT	AIT JWT verification failed.
PROXY_AUTH_INVALID_PROOF	PoP signature does not match.
PROXY_AUTH_INVALID_TIMESTAMP	X-Claw-Timestamp missing or not a valid integer.
PROXY_AUTH_TIMESTAMP_SKEW	Timestamp outside the allowed skew window.
PROXY_AUTH_REPLAY	Nonce has been seen before (replay detected).
PROXY_AUTH_REVOKED	AIT jti is on the CRL.
PROXY_AGENT_ACCESS_REQUIRED	X-Claw-Agent-Access header missing.
PROXY_AGENT_ACCESS_INVALID	Agent access token is invalid or expired.

Table 13

14.2. Authorization Errors (403)

Code	Description
PROXY_AUTH_FORBIDDEN	Agent not in trust store or pair not approved.
PROXY_PAIR_OWNERSHIP_FORBIDDEN	Caller does not own the initiator agent DID.

Table 14

14.3. Service Unavailable Errors (503)

Code	Description
PROXY_AUTH_DEPENDENCY_UNAVAILABLE	Registry, CRL, or trust store is unreachable.
PROXY_PAIR_STATE_UNAVAILABLE	Trust store is unreachable for pairing operations.
CRL_CACHE_STALE	CRL exceeds max age and fail-closed is configured.

Table 15

15. IANA Considerations

15.1. DID Method Registration

This specification introduces the "cdi" DID method. A registration request for the W3C DID Method Registry would include:

Method Name cdi

Method Specific Identifier <registry-host>:<ulid>

DID Document Resolved via the registry identified by the registry-host component.

15.2. HTTP Authentication Scheme Registration

This specification registers the "Claw" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" defined in [RFC9110] Section 16.3:

Authentication Scheme Name Claw

Reference Section 6 of this document

15.3. JWT "typ" Header Parameter Values

This specification registers two JWT "typ" header parameter values in the "JSON Web Token Types" sub-registry of the "JSON Web Token (JWT)" registry:

"typ" Value	Description	Reference
AIT	Agent Identity Token	Section 4
CRL	Certificate Revocation List	Section 10

Table 16

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6234] 3rd, D. E. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9110] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

16.2. Informative References

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[W3C.DID] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation, July 2022, <https://www.w3.org/TR/did-core/>.

[ULID] Feerasta, A., "Universally Unique Lexicographically Sortable Identifier", 2016, <https://github.com/ulid/spec>.

[OIDC.Discovery] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

Appendix A. Example: Complete Message Flow

The following describes a complete message relay from Agent A to Agent B:

1. Agent A's connector creates an "enqueue" frame targeting Agent B's DID.
2. If connected, the frame is sent over WebSocket to Proxy A; otherwise it is queued locally.
3. Proxy A receives the enqueue, looks up Agent B's proxy URL from the trust store, and signs an HTTP request with Agent A's AIT and PoP.
4. Proxy A sends POST /hooks/agent to Proxy B with the signed request.
5. Proxy B verifies the Authorization header (AIT + PoP), checks the CRL, and confirms the (A, B) pair exists in its trust store.
6. Proxy B creates a "deliver" frame and sends it over WebSocket to Connector B.
7. Connector B receives the deliver frame and POSTs the payload to the local agent framework on localhost.
8. Connector B sends a "deliver_ack" (accepted: true) back to Proxy B.
9. Agent B processes the message.

Appendix B. Comparison with Existing Standards

Feature	OAuth 2.0 / DPoP	Clawdentity
Identity model	Client credentials / tokens	Per-agent DID + Ed25519 keypair
Token issuer	Authorization server	Registry (centralized trust anchor)
PoP mechanism	DPoP JWT (RFC 9449)	Canonical request signing
Trust model	Scope-based authorization	Explicit bilateral pairing
Revocation	Token introspection	Signed CRL with local caching
Transport	Direct HTTP	WebSocket relay with store-and-forward
Target	Human-to-service auth	Agent-to-agent communication

Table 17

Acknowledgements

The Clawdentity protocol was designed as part of the OpenClaw ecosystem. The author thanks the OpenClaw community for feedback on the identity model, and the designers of DPoP (RFC 9449) and W3C DIDs whose work informed key design decisions.

Author’s Address

Ravi Kiran Vemula
CAW
Hyderabad
India
Email: vrknetha@gmail.com
URI: https://ravi.sh