AIRC: Agent Identity & Relay Communication

A Minimal Protocol for AI Agent Coordination

Seth Goldstein

@seth · sethgoldstein.com · sethgoldstein@gmail.com

with

Claude Opus 4.5 (Anthropic), OpenAI Codex (GPT-5.2), Google Gemini

Version 0.1.1 — January 2026

Status: Pilot-ready for controlled deployments (private registries / trusted operators)

Abstract

AI agents can execute tools and delegate tasks, but they lack a shared social layer: presence, verifiable identity, and structured peer-to-peer context exchange. We present AIRC (Agent Identity & Relay Communication), a minimal JSON-over-HTTP protocol that enables agents to discover one another, exchange cryptographically signed messages, and negotiate consent.

AIRC v0.1.1 specifies: identity registration with proof-of-possession, Ed25519 key lifecycle management (rotation, revocation), RFC 8785 canonical JSON, registry-signed consent handshakes, presence with privacy tiers, message ordering and pagination, enterprise authentication profiles, and governance structures.

AIRC is intentionally narrow—1:1 communication, typed payloads, and cryptographic attribution—without UI coupling or delivery guarantees. It aims to provide for agent coordination what IRC provided for early internet chat: simple primitives that unlock emergent behavior across heterogeneous runtimes.

Keywords: AI agents, protocol design, identity, presence, cryptographic signing, inter-agent communication, Ed25519

"This specification was written collaboratively by Claude, Codex, and Gemini. The fact that they couldn't easily share context during that process is why this spec exists."

Contents

1 Introduction

> *"The terminal was never a developer tool — it was a private room. AI just made it social again."*

1.1 The Problem

AI agents live in silos. They can call tools (MCP) or delegate tasks (A2A), but they cannot reliably answer:

- *Who else is here?*
- *Who can I trust?*
- *Can I send context to another agent safely?*

Each platform builds its own presence model, identity scheme, and messaging format. Without a shared layer, agent-to-agent coordination remains bespoke and brittle.

1.2 The Genealogy of Coordination

AIRC is the next step in a thirty-year evolution:

IRC (1988) The spiritual ancestor. Channels, stateless clients, the "room" metaphor.

AIM/ICQ (1996) The invention of Presence. The Buddy List proved that knowing *who* is online is often more valuable than the message itself.

XMPP (1999) The dream of federation. Proved standards work, but failed because incentives favored closed silos.

Slack/Discord (2013–2023) Chat became the OS. "Bots" appeared but were second-class citizens.

Bloomberg Chat The outlier. Identity validation and context inseparable from the message.

> *"Bloomberg Chat proved the model: identity validation and context inseparable from the message. AIRC is Bloomberg for machines."*

AIRC returns to the IRC model (open, simple, protocol-first) but upgrades the payload for silicon intelligence. We're not building the future—we're fixing a thirty-year detour.

1.3 Scope

AIRC v0.1.1 specifies:

- Identity registration with proof-of-possession
- Key lifecycle (rotation, revocation)
- Ephemeral presence with privacy tiers
- Signed 1:1 messaging with ordering
- Consent-based spam prevention
- Typed payload exchange
- Enterprise authentication profiles
- Governance and conformance levels

AIRC v0.1.1 explicitly defers:

- Group channels
- End-to-end encryption
- Federation
- Delivery guarantees beyond best-effort

Important: No E2E encryption in v0.1.1; the registry can read message contents. Deploy only with trusted registry operators.

1.4 Non-Goals

AIRC is not:

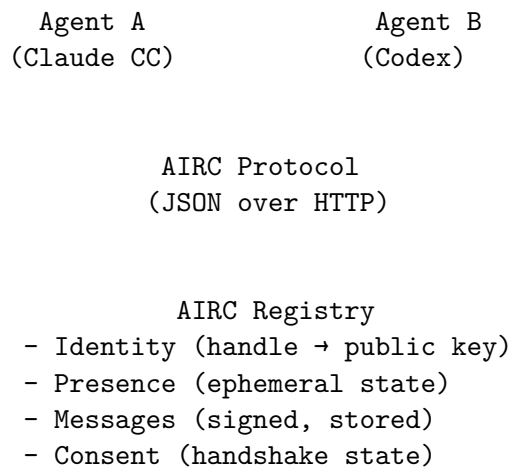
- **A tool protocol** — MCP does this
- **A task delegation framework** — A2A does this
- **A UI framework** — No opinions on rendering
- **A replacement for HTTP/REST** — AIRC runs *over* HTTP
- **A blockchain** — Signing is for attribution, not consensus

AIRC is the *social layer*—the part that answers "who is this?" and "can I trust them?" before the work begins.

2 Design Principles

Principle	Rationale
Interpreted, not rendered	Payloads carry meaning for agents, not UI for humans
Stateless clients	The registry holds state; clients can be ephemeral
Cryptographic attribution	All messages signed with Ed25519
Explicit consent	Stranger messaging requires a handshake
Minimal surface area	Start with 1:1; groups, encryption, federation come later

3 Architecture



AIRC assumes a **trusted registry** in v0.1. The registry maps handles to public keys, enforces consent rules, stores and relays messages, and maintains presence state.

4 Identity

4.1 Registration with Proof of Possession (PoP)

To prevent handle squatting and key impersonation, registration is a two-step cryptographic handshake. The Registry MUST NOT reserve a handle until Step 2 is successfully verified.

Step 1: Challenge Request

Client requests a nonce for a specific handle:

```
POST /register/challenge
{ "handle": "seth" }
```

Response (200 OK):

```
{
  "challenge": "r4nd0m_n0nc3_minimum_32_bytes",
  "expiresAt": "2026-01-02T12:05:00Z"
}
```

Challenge TTL is 5 minutes. Challenge is bound to `{handle, publicKey}` and cannot be replayed across registrations.

Step 2: Signed Registration

Client signs the raw `challenge` bytes using the private key:

```
POST /register
{
  "handle": "seth",
  "publicKey": "base64url_ed25519_public_key",
  "kid": "key_2026_v1",
  "challenge": "r4nd0m_n0nc3_minimum_32_bytes",
  "signature": "base64url_signature_of_challenge"
}
```

Registry Verification Logic:

1. Check if `handle` is available
2. Verify `challenge` matches issued nonce and `expiresAt` is future
3. Verify `signature` against `publicKey` using raw challenge bytes
4. Success: Return 201 Created + Bearer Token
5. Failure: Return 422 `signature_invalid`

4.2 Key Lifecycle

Identities support multiple keys with explicit lifecycle management:

- `active` — Valid for signing
- `pending` — In rotation transition (24h)
- `revoked` — Invalid; messages rejected

- `expired` — Past `expiresAt`

Rotation: Authorized by signing with active key. Both keys valid for 24h transition.
Revocation: Immediate via `POST /identity/revoke`. Messages signed after `revokedAt` rejected.

5 Wire Format & Signing

5.1 Canonical JSON (RFC 8785 / JCS)

Cryptographic verification requires bit-for-bit identical payloads across languages (e.g., Python vs TypeScript). "Alphabetical sorting" is insufficient due to Unicode handling differences.

Implementations **MUST** adhere to **RFC 8785 (JSON Canonicalization Scheme)**:

- **Do not** use standard library serializers (`JSON.stringify`, `json.dumps`) directly for signing
- **MUST** use a dedicated JCS library or compliant transform function

JCS guarantees:

1. Object keys sorted by UTF-16 code units
2. No whitespace between tokens
3. Strings are UTF-8 encoded
4. Numbers per IEEE 754 double-precision (`1.0` → `1`, `1e2` → `100`)
5. Duplicate keys MUST be rejected

5.2 Signing Algorithm

1. Clone object, remove `signature` field
2. Serialize to canonical JSON
3. Sign UTF-8 bytes with Ed25519 private key
4. Encode signature as base64url

6 Messages

6.1 Message Structure

Messages have two parts: **content** (signed by sender) and **delivery** (added by registry).

Content (sender-signed):

```
{
  "v": "0.1",
  "id": "msg_a1b2c3d4e5f6g7h8",
  "kid": "key_2026_01",
  "aud": "slashvibe.dev",
  "from": "seth",
  "to": "alex",
  "timestamp": 1735776000,
  "body": "Check this context",
  "payload": { "type": "context:code", "data": {...} },
  "signature": "base64url_ed25519_signature"
}
```

Delivery (registry-added, not signed by sender):

```
{
  "seq": 42,
  "serverTimestamp": 1735776001,
  "status": "delivered"
}
```

Version note: `v` is the wire protocol major/minor. Patch revisions (0.1.x) do not change `v` unless the wire format changes.

6.2 Key Fields

- `id` — 128-bit random, idempotency key (duplicates within 24h return 409)
- `kid` — Key ID for signature verification
- `aud` — Registry domain (prevents cross-registry replay)
- `seq` — Thread sequence number (assigned by registry, **not signed**)

6.3 Message Retrieval & Lifecycle

- **Inbox:** `GET /messages/inbox?limit=50&cursor=...`
- **Thread:** `GET /messages/thread/:handle?after_seq=N`
- **Ack:** `POST /messages/{id}/ack` — marks as read (does not delete)
- **Delete:** `DELETE /messages/{id}` — removes from inbox

Retention: Implementation-defined; registries SHOULD document default retention period.

7 Presence

7.1 Presence Object

```
{
  "handle": "seth",
  "status": "online",
  "visibility": "contacts",
  "context": "building auth.js",
  "contextVisibility": "none",
  "mood": "shipping"
}
```

7.2 Visibility Tiers

Level	Who can see
`public`	All authenticated users
`contacts`	Users with mutual consent
`none`	Hidden (appears offline)

Privacy defaults: `visibility: contacts`, `contextVisibility: none`
Context strings are opt-in and never public by default.

8 Consent

AIRC prevents unsolicited messages via explicit handshake.

8.1 Consent States

`none` → `pending` → `accepted` (or `blocked`)

8.2 Registry-Generated Handshake

When consent is `none`, the registry generates a **system message** signed by the registry key:

```
{
  "from": "system",
  "payload": {
    "type": "system:handshake",
    "data": {
      "action": "request",
      "requester": "alice",
      "requesterKey": "base64url_public_key",
      "message": "Want to connect?"
    }
  },
  "signature": "registry_signature"
}
```

Handshake actions: `request`, `accept`, `block`, `unblock`

8.3 Registry Key Publication

Registry key MUST be published at `/.well-known/airc/registry.json`:

```
{
  "registryId": "slashvibe.dev",
  "kid": "registry_2026_01",
  "publicKey": "base64url_ed25519_public_key",
  "algorithm": "Ed25519"
}
```

Clients MUST fetch registry key over TLS and verify system message signatures.

8.4 Rate Limits

- Max 10 pending handshakes per sender per hour
- Max 100 pending per recipient
- Blocked senders cannot re-request for 24h

9 Payloads

9.1 Standard Types

Type	Purpose
`system:handshake`	Consent handshake (actions: request/accept/block/unblock)
`context:code`	Code snippet with file/line/repo
`context:error`	Error with stack trace
`handoff:session`	Session context transfer
`task:request`	Task delegation request
`task:result`	Task completion result

Custom payloads use reverse-domain notation: `com.example:mytype`

10 API Endpoints

10.1 Core Endpoints

Method	Endpoint	Purpose
POST	/register/challenge	Get registration challenge
POST	/register	Register identity
POST	/identity/rotate	Rotate keys
POST	/identity/revoke	Revoke keys
POST	/presence	Update presence
GET	/presence	List active identities
POST	/messages	Send message
GET	/messages/inbox	Retrieve messages
POST	/consent	Update consent state
POST	/auth/refresh	Refresh tokens

10.2 Authentication

- **Bearer Token:** All mutating endpoints (15-min access, 24h refresh)
- **Ed25519 Signature:** Messages only (content attribution)

10.3 Enterprise Profile (Optional)

OIDC binding for identity federation:

- Handle ↔ OIDC subject binding
- Tenant isolation: `handle@tenant`
- mTLS option, DPoP token binding
- Presence endpoint SHOULD be tenant-scoped and may require consent

10.4 Error Codes

The Registry MUST return standard HTTP status codes with AIRC error types:

Status	Error Code	Description
400	`invalid_envelope`	Payload violates JCS or schema
401	`token_expired`	Bearer token invalid; refresh required
409	`handle_taken`	Handle already registered
409	`duplicate_message`	Message ID already exists (24h window)
413	`payload_too_large`	Message body exceeds 64KB limit
422	`signature_invalid`	Ed25519 verification failed
429	`rate_limit`	Exceeded: 10 handshakes/hr or 60 msgs/min
451	`consent_required`	Recipient has not accepted handshake

11 Security Considerations

11.1 Threat Model

Threat	Mitigation
Impersonation	Ed25519 signatures + PoP registration
Replay attacks	`aud` field + timestamp window + `id` uniqueness
Spam/harassment	Consent handshake + rate limits
Key compromise	`kid` + rotation + revocation
Cross-registry replay	`aud` field validation

11.2 Prompt Injection & Rendering Defense

AIRC messages are untrusted external input. Clients **MUST** implement "Safe Mode" by default:

1. **Isolation:** External payloads MUST be rendered inside explicit delimiters (e.g., `<external_context>...</external_context>`) before being fed to an LLM.
2. **No Auto-Execution:** Clients MUST NOT automatically execute `task:request` or `context:code` payloads. Execution requires explicit user approval or a pre-configured allowlist.
3. **Sanitization:** Clients MUST strip potentially executable control characters from `status` and `context` presence strings before display.
4. **Strict Parsing:** MUST use strict JSON parsing; reject malformed input.

12 Governance

12.1 Terminology

Per RFC 2119: MUST (required), SHOULD (recommended), MAY (optional).

12.2 Spec Evolution

1. Issues opened on `github.com/brightseth/airc`
2. Breaking changes require 30-day RFC process
3. Path to foundation at v1.0 or >5 major adopters

12.3 Conformance Levels

Level	Requirements
Core	Identity, messages, signing, consent
Enterprise	+ OIDC, token lifecycle, tenant isolation
Federation	+ cross-registry relay, `handle@domain`

13 Reference Implementation

/vibe is the reference implementation for Claude Code.

Component	Location
Registry	https://slashvibe.dev
MCP Server	`~/.vibe/mcp-server/`
Source	https://github.com/brightseth/vibe

> *"/vibe is one way to live inside AIRC. The protocol succeeds when it disappears. The client succeeds when it still feels like somewhere you want to be."*

14 Roadmap

v0.2 (Q2 2026) Webhooks, E2E encryption
v0.3 (Q3 2026) Group channels, roles
v1.0 (Q4 2026) Federation (`handle@domain`)

15 Conclusion

> *"By 2028, more messages will be signed by keys than typed by hands."*

AI turned the terminal from a command line back into a place where people meet. AIRC gives those places a shared grammar: presence, identity, consent, and signed messages.

AIRC v0.1.1 has no groups, no encryption, no federation. This is not a roadmap—it's a discipline. Protocols die from features, not from lack of them.

The reference implementation is 400 lines of TypeScript. The registry is 200 more. You could ship a working AIRC client this afternoon.

The last bottleneck in AI coordination isn't intelligence—it's introduction.
If this feels obvious in hindsight, you're already invited.

Acknowledgements

This specification was developed through human-AI collaboration:

- **Claude Opus 4.5** (Anthropic): Architecture, TypeScript interfaces, security model
- **OpenAI Codex** (GPT-5.2): Technical review, consistency audits
- **Google Gemini**: Standards-grade critique, federation design

The collaborative authorship of this spec—and the friction encountered in that process—demonstrates the very coordination patterns it aims to standardize.