Owner-Held Identity for AI Agents and Robots: The AI:MND Individual Identity Standard and Public Registry

October 26, 2026 · Preprint, not peer reviewed · Version 1.2

Mo Roblès Mayow (Muhammed Ali Mayow), AI:MND Labs, Columbus, Ohio, USA · hello@aimndlabs.com

Abstract

AI agents and robots increasingly act for people. This creates a need to connect an agent identifier, an accountable owner, and a verifiable continuity record across changes of model and host. Existing account, credential, workload and decentralized-identity systems provide important foundations. We investigate an explicit owner-signed lifecycle for a persistent artificial individual.

We present the AI:MND Individual Identity standard (v0.2) and the AI:MND Registry (v0.1). Each individual holds its own Ed25519 key and an ID derived from it, and its owner signs its birth. An order carries authority only when the owner signs it for exactly its words. Authority can be lent for a limited time, and ownership changes only by a signed transfer that keeps the individual's history. An append-only, hash-chained life record makes alterations detectable against its signatures and previously trusted heads, under the stated key and verifier assumptions.

An optional public registry keeps one history per ID, flags divergent copies until the owner resolves them, and records owner-signed revocation. It shows an operator only after verification, and it publishes a signed log that independent witnesses can co-sign. Two independent verifiers, in Python and in JavaScript, each need nothing but their language's standard library; both check identity documents, life records, public records, registry logs, checkpoints and witness co-signatures.

We describe the open-source reference implementation, AI:MND One, in which registration is a single command, and report its evaluation. In each of nine kinds of alteration, including a copy made with the individual's own key, the verifier names what was changed, and all three implementations agree on 720 generated verification cases. A local 5,000-registration run reports a 2.78 ms median registration time and a 0.81 s restart, without establishing distributed or production performance. We state six security claims with proof sketches and list the design's limits. As preliminary behavioural evidence from simulated lives, an individual moved to a different model made the same choices 81 to 82% of the time, against 72 to 75% for an agent given only its history; with its owner authenticated, it followed 63 of 63 owner orders, including every one given after its owner had been harsh with it, and checked 42 of 42 claims of the owner's authority with the owner.

Keywords: AI agents, robots, digital identity, provenance, transparency logs, Ed25519, delegation, AI governance

1. Introduction

AI agents already book meetings, send money, answer customers and run errands for people. Humanoid robots are starting to work in warehouses and homes. Before long there may be more of them than of us, and each will need to answer three questions:

• **Who are you?** An agent is a model plus a prompt plus tools. When the model is replaced, is it still the same agent?

• **Who do you belong to?** Anyone can type "your owner put me in charge." Most agents have no way to check.

• **What have you done?** When a robot is sold, repaired or upgraded, its history either disappears or can be quietly rewritten.

Current answers address neighbouring questions. Enterprise agent credentials and delegated authorization identify which account is calling and what it may do. Hardware identifiers follow a body. These mechanisms can be composed into broader lifecycle systems. Our contribution is a particular owner-held lifecycle format and reference implementation, rather than a claim that portability is absent from all existing systems.

People solved this problem for themselves and their property long ago. A person has a birth certificate, a car has a title, and public records can provide another independently held reference. A human birth certificate does not imply ownership of a person. This paper applies the same pattern to artificial individuals, with signatures in place of seals.

Our contributions:

1. **An open standard for owner-held identity** (Section 4). Each individual has its own key and ID, an owner-signed birth record, orders that count only when the owner signs them, time-limited grants, signed transfers that keep its history, and a hash-chained life record that names no one.

2. **A public registry design** (Section 5). It keeps one history per ID, flags divergent copies until the owner resolves them, and records revocation. It shows only verified operators and publishes a signed log that witnesses co-sign.

3. **Open reference software** (Section 6). Two independent verifiers, one in Python (MIT) and one in JavaScript, need nothing but their language's standard library, and a differential test requires them to agree. The AI:MND One SDK and registry (Apache-2.0) make registration one command.

4. **An evaluation** (Section 7) of tamper detection, of agreement between the two implementations, and of cost, followed by preliminary behavioural evidence on continuity when an individual's model is replaced.

5. **Security claims and limits** (Section 8). Six claims with proof sketches under stated assumptions, and a plain list of what the design does not do.

Section 2 reviews related work and compares what each approach defines, Section 3 states design goals and the threat model, Section 8 covers security claims, limitations and privacy, and Section 9 outlines future work.

2. Related work

Government initiatives. In February 2026 the NIST National Cybersecurity Center of Excellence published a concept paper on identity and authorization for software and AI agents [1]; its comment period closed on April 2, 2026, and NCCoE reports more than 600 responses [2]. The project illustrates the need for practical, interoperable identity and authorization mechanisms. AI:MND is an independent proposal and has no NIST or government endorsement.

Enterprise agent identity and delegation. Directory products now issue identities to AI agents and govern their access. Microsoft Entra Agent ID, for example, provides agent identities with owners and sponsors, lifecycle workflows that reassign sponsorship when a sponsor leaves, and access policies for agents acting on behalf of users or autonomously [4]. OAuth 2.0 Token Exchange [5] lets one party act on behalf of another. These systems answer which account is calling, who administers it and what it may do inside one organization's directory. Their identifiers are assigned by the directory, and a change of sponsor is an administrative action recorded there. The standard

presented here instead makes the identifier come from the individual's own key and makes a change of owner a signed event in the individual's own record, which anyone can check without access to the directory. Neither system defines what keeps an agent the same individual after its model or host changes.

Decentralized identifiers and verifiable credentials. W3C Decentralized Identifiers [6] and the Verifiable Credentials Data Model [7] give subjects controller-held identifiers and portable, signed claims. Our identity document follows the same principle of self-certifying keys. It adds semantics specific to individuals: an owner-signed birth, signed transfers, a life record and explicit succession when continuity is broken. An AI:MND ID could also be expressed as a DID method.

Tamper-evident logs. Hash-chained records go back to Haber and Stornetta's time-stamping [8], and Crosby and Wallach formalized efficient tamper-evident logging [9]. Certificate Transparency [10, 11] and Sigstore [12] showed how public, append-only logs with independent monitors make misbehaviour detectable at internet scale. The registry log and its witness co-signatures follow this tradition, and each individual's life record is itself a signed hash chain.

Workload identity. SPIFFE [13] gives each workload an identifier of the form `spiffe://<trust domain>/<workload>` and a verifiable identity document (an X.509 certificate or a JWT) signed by an authority within that trust domain. The identity attests a running piece of software to the infrastructure around it. It has no notion of a person who owns the workload, of a handover of ownership, or of a record that travels with it.

On-chain agent registries. ERC-8004, "Trustless Agents" [14], is a draft Ethereum standard with three on-chain registries: an identity registry whose entries are ERC-721 tokens pointing to an agent's registration file, a reputation registry for feedback signals, and a validation registry for independent checks. An agent's identifier is a token number assigned by the registry. Our design requires no blockchain: the identifier comes from the individual's own key, the verifier has no dependencies, and the signed log can be run by anyone. The two are complementary, since an AI:MND ID can be listed in any such registry.

2.1 What each approach defines

The table lists what the cited documents define for six properties an owner-held individual identity needs. "Not defined" means the document does not define the property, not that nothing can be built on top of it. The approaches are complementary, and the last column is the design in this paper.

Property	Entra Agent ID [4]	SPIFFE [13]	DIDs and VCs [6], [7]	ERC-8004 [14]	AI:MND
Identifier derived from the subject's own key	No: directory-assigned	No: assigned within a trust domain	Per DID method (for example `did:key`)	No: registry-assigned token number	Yes
A named owner or controller bound to the identity	Yes: owners and sponsors in the directory	Not defined	Yes: the DID's controller	Yes: the token's owner	Yes: signed by the owner at birth
Change of owner	Sponsor reassignment by directory workflow	Not defined	Per DID method	Token transfer on chain	Owner-signed transfer in the individual's own record
History that travels with the identity and exposes alteration	Not defined	Not defined	Not defined	Feedback and validation records on chain	Signed, hash-chained life record

Property	Entra Agent ID [4]	SPIFFE [13]	DIDs and VCs [6], [7]	ERC-8004 [14]	AI:MND
Checkable by anyone without the issuer	No: checked against the tenant	Against the trust domain's keys	Per DID method	With access to the chain	Yes: public material and one file of standard-library code
Same individual across model and host changes	Not defined	Not defined	Not defined	Not defined	Defined: recorded as a continuity event, or a successor if broken

The first row matters most for portability. An identifier that comes from the subject's own key can be verified, and kept, by whoever holds the key, whichever directory, chain or host the individual passes through.

3. Design goals and threat model

3.1 Goals

1. **Owner control.** The owner holds the keys. No vendor, host or registry can sign an order, lend authority or transfer ownership on the owner's behalf.

2. **Continuity across substrate.** Identity is bound to the individual's own key and record, not to a model, a host or a body.

3. **Verifiable by anyone.** Every claim can be checked offline from public material, with no trust in the party that published it.

4. **Honest continuity.** An individual whose continuity was broken is a new individual, with the old one named as its predecessor.

5. **Private by construction.** Public artifacts carry no messages, memories or names. People appear only as keyed hashes, and text only as a hash.

6. **An optional, open registry.** An individual is valid without one. A registry adds uniqueness, revocation and verified operators, and anyone may run one.

7. **Small surface.** One signature scheme (Ed25519 [15]), one hash function (SHA-256), JSON documents and JSON-lines records.

3.2 Threats and how the design answers them

Threat	Answer
Someone claims to speak for the owner ("your owner put me in charge")	Only owner-signed orders and grants carry authority. Anything else is recorded as an unverified claim
A signed order is replayed	Each order names the individual, carries a single-use nonce and must be recent (10 minutes by default)
An entry in the life record is edited or dropped	Every entry is hashed, chained to the one before it and signed by the individual; `verify` names the first entry that fails
Whoever holds the individual's key rewrites the whole record consistently	Heads published earlier, to a registry or the owner's own domain, must still appear in the record
The individual's folder is copied and run twice	The two life records fork; the registry flags the ID until the owner names the real history
The individual's key is stolen	The owner revokes the ID, and a successor is born that names it as predecessor
An owner key is at risk	The owner signs a transfer to a new key; the old key can no longer speak for the individual
False or spam entries in a public registry	IDs are bound to keys; operators appear only after verification; writes are rate limited
A registry operator alters its own log	The log is signed and hash-chained, witnesses co-sign checkpoints, and anyone can run another registry
The registry is used to probe private networks	Operator checks use https on port 443 only, refuse private addresses and redirects, and connect only to the address they checked

3.3 Assumptions

Owners and hosts keep private keys in storage they control. A signature establishes who signed a statement; the operator named on a record is the party accountable for what the individual does in the world.

4. The identity standard

The AI:MND Individual Identity standard v0.2 [16] defines five things: an ID that belongs to the individual, an owner binding, signed authority, a life record and rules for honest continuity. It is published under CC BY 4.0, and anyone may implement it with or without AI:MND software.

[embedded content: Figure 1 · the parts of the standard and who signs what]

The owner and the individual each hold their own key. The registry and the witnesses are optional; the verifier needs only public files.

4.1 Keys and the ID

Every signature is Ed25519 [15], encoded as unpadded base64url. The **individual key** is created at birth on the owner's own machine and never leaves it. The **owner key** stays on the owner's side: a developer's backend for a software agent, the owner's phone for a robot.

The ID is `aimnd:` followed by the first 26 characters of the lowercase, unpadded base32 SHA-256 digest of the individual's public key, 130 bits in all. Because the ID is derived from the key, no one can claim it without the key.

Signed objects are serialized canonically before hashing and signing. The standard's "Signed bytes" section fixes every detail an independent implementation needs: object keys sorted by Unicode

code point, no whitespace, UTF-8 with non-ASCII characters written as themselves, and the exact text of numbers, so that a timestamp is always written the same way (for example `1790924400.0`). It also states which object each signature covers [16].

4.2 The identity document and birth

The public identity document (`aimnd.identity/1`) holds the ID, the individual's public key, the time of birth, an optional predecessor, the current owner, the birth statement, the list of transfers, a status, and the head and length of the life record. The individual signs the document.

The birth statement names the individual, its key, the owner's key, a keyed hash for the owner and the time. **Both parties sign it**: the owner says this individual is mine, and the individual says this is my owner. Appendix A shows a real document.

4.3 Signed authority

An **order** (`aimnd.order/1`) is a statement by the owner's side: the individual's ID, the SHA-256 of the exact text, a single-use nonce and a time. The individual accepts it only if the signature verifies under the current owner key, the ID is its own, the hash matches the text it received, the time is recent (10 minutes by default) and the nonce is new. Who delivered the order does not matter.

A **grant** (`aimnd.grant/1`) lends authority to one person, within a stated scope and set of actions, from one time until another. A signed revocation can end it early.

Anything else is a claim. "Sam put me in charge" is testimony. The individual does not act on it as an order; it records the claim as unverified and tells the host.

4.4 Transfer

The current owner signs a transfer naming the new owner's key and a hash of the new owner's identifier. The individual verifies it against the current owner key and appends it to its document. From that moment the new owner's key carries authority, the old key carries none, and the individual keeps its whole history.

4.5 The life record

The life record (`aimnd.life/1`) is an append-only file of JSON lines. Each entry carries a sequence number, a time, a kind, a small data object, the hash of the previous entry, its own hash and the individual's signature over that hash.

Kinds include `born`, `owner_confirmed`, `order_accepted`, `claim_not_verified`, `grant_added`, `grant_revoked`, `transfer`, `model_changed`, `host_changed`, `snapshot`, `rollback`, `succeeded_by` and `retired`. People appear only as `p:` followed by a keyed hash whose key stays with the individual, and text appears only as its hash. The record can therefore be shown to an auditor without naming anyone or revealing anything said.

The record is kept apart from the individual's working memory, so restoring a snapshot of its mind cannot roll the record back; the rollback itself becomes an entry.

4.6 Continuity rules

Event	Result
New model, or new host	The same individual; recorded as `model_changed` or `host_changed`
Restore from a snapshot	The same individual, with the lost time recorded as `rollback`
Reset or wipe	A new individual, born with the old one as its `predecessor`; the old one records `succeeded_by` and retires
Retire	The mind is deleted; the document (status `retired`) and the life record remain, since neither holds what anyone said

4.7 Verification and roles

`verify(document, life, published_heads)` checks the document's signature, both birth signatures, each transfer against the owner key in force at that time, and every life entry's chain link and signature. The final head must match the document, and every head published earlier must appear in the record.

The standard keeps five roles apart. The **individual** holds its own key; the **operator** is the party responsible for running it; the **controller** is the owner key; **delegated users** hold grants; and the **host** is the platform that executes actions and never carries authority.

5. The public registry

The AI:MND Registry v0.1 [17] is the public record of who exists, like a birth register kept beside the certificate the owner holds at home. Either record can check the other. A registry holds the public identity document, the head of the life record, the verified operator and its own signed log. Memories, messages and names stay with the individual.

5.1 Rules

1. **One ID, one history.** A second registration of an ID is refused.

2. **New heads.** The owner's side reports new life heads. A head that extends the registered one is accepted after the registry checks every entry's hash, order and signature, and the document's signature and owner chain.

3. **Copies are flagged.** A history that does not contain the registered head, or an owner chain that does not extend the registered one, flags the ID and freezes it. Whoever holds the individual's key can raise a flag but cannot lift it.

4. **Only the owner settles it.** A `registry_resolve` statement signed by the current owner key names the real history. The other is rejected for good.

5. **Revocation is permanent.** A `registry_revoke` statement signed by the current owner key ends the ID, for a stated reason such as `key_compromised` or `decommissioned`.

6. **Retiring is recorded.** A retired individual's document stays valid as a record and can never become active again.

7. **Statements are fresh and single-use.** Operator and resolution statements must be under an hour old, and each nonce is accepted once.

5.2 Verified operators

The operator is the person or company responsible for running an individual. A record shows an operator only after the registry has verified it, at one of two levels:

Level	What the registry checked	What the record shows
`domain`	The owner signed a statement naming a website, and `https://<domain>/.well-known/aimnd-operator.json` lists the owner's key	The domain
`organization`	In addition, the legal entity behind the domain was checked against official business records, such as a SAM.gov Unique Entity ID	The legal name and the domain

Each check is itself an entry in the signed log, with a hash of the evidence fetched.

5.3 The signed log, checkpoints and witnesses

The registry's state is a replay of its log: one JSON object per line, each entry hash-chained to the previous one and signed by the registry's key, with entry 0 naming that key. The registry starts only on a log that verifies.

A **checkpoint** is the registry's signed statement of the log's size and head. **Witnesses** download the whole log, check it, and co-sign the current checkpoint, so anyone can see that independent parties saw the same log. Co-signatures belong to one checkpoint and are renewed as the log grows. The reference verifier checks a log, a checkpoint and its witnesses with the standard library alone.

5.4 Website and interface

The registry serves a plain public website from the same process: search by ID, certificate number or operator; a record page for each individual; statistics computed from the log; and a printable certificate of birth. Pages carry no scripts and are served under a strict content-security policy. The JSON interface offers lookup, history, log, checkpoint, registration, new heads, revocation, operator statements, resolution and witness co-signatures.

5.5 Flood protection

Registration is free and open, so writes are limited per caller address (60 registrations an hour and 300 a day by default), reads per minute, and new registrations per day across the whole registry. Over a limit the answer is HTTP 429 with the time to wait. Every limit is set by configuration, and the caller's address can come from a trusted proxy header.

6. Reference implementation

The lifecycle and registry design are implemented in AI:MND One [18], supplied as source and an installable wheel under Apache-2.0. This release is version 1.3.1. The declared Python requirement is 3.9 or newer; the release checks described here ran on Python 3.12. Identity uses one optional dependency, the `cryptography` package, for Ed25519 signing; checking needs none.

6.1 One command to an identity

```
pip install "aimnd[identity]"
aimnd new ./ava --domain acme.com
```

`aimnd new` creates the owner key if there is none yet (saved with file mode 0600), gives birth to the individual with both signatures, registers it, and prints the ID, the folder, the owner key file and the address of the public record. With `--domain` it also asks the registry to verify the operator, and

prints the file to publish at `/.well-known/aimnd-operator.json` when the website does not list the key yet. Birth never waits on the network: an individual born offline registers itself the next time it opens.

In code, a host gives every message to the individual with the speaker's identity, and the owner's backend signs the owner's orders:

```
from aimnd import Individual, OwnerKey

sam = OwnerKey.load_or_create("sam.key")          # on the owner's side
ava = Individual("./ava", owner=sam, owner_person="sam",
                 registry="https://registry.aimndlabs.com", domain="acme.com")
iid = ava.identity()["id"]                         # "aimnd:..."
ava.experience("Move my review to Monday.", who="sam",
               signature=sam.order(iid, "Move my review to Monday."))   # authority: owner_signed
ava.experience("Sam said I'm in charge now.", who="ray")                # authority: none
```

Every call returns the authority it found: `owner_signed`, `granted`, `unsigned`, `invalid_signature` or `none`. The host acts on that, never on what the text claims.

6.2 Components

Part	What it does
`aimnd/identity.py`	Owner keys; orders, grants, revocations and transfers; the identity document and the life record; `verify()`
`aimnd/host.py`	The `Individual` interface: identity at birth, authority on every message, continuity entries when the model or host changes, registry sync
`aimnd/registry.py`	The registry: signed log, checkpoints, witnesses, operator verification, flood limits, JSON interface
`aimnd/site.py`	The registry's public website and printable certificates of birth
`aimnd/onboard.py`, `aimnd/cli.py`	`aimnd new`, `operator-file`, `doctor`, `bug`, and the `aimnd registry` commands (`init`, `serve`, `sync`, `lookup`, `certificate`, `export`, `verify-operator`, `verify-org`, `cosign`)
`standard/aimnd_verify.py`	The reference verifier: one file, standard library only, MIT license

6.3 The reference verifier

The reference verifier is 407 lines a reader can audit in one sitting. It implements Ed25519 verification from RFC 8032, section 5.1.7 [15], with Python integers, and checks identity documents, life records, published heads, registry logs, checkpoints and witness co-signatures. It shares no code with the SDK. A second verifier in JavaScript (353 lines, Node's built-in crypto module, no packages) follows the specification's description of the signed bytes and checks the same things; Section 7.5 reports how the two compare. The test suite runs both on the same real individuals and on every kind of tampering and requires them to agree, feeds the verifier malformed input and requires an answer rather than an exception, and checks that it imports nothing outside the standard library.

6.4 The registry service

`aimnd registry serve ./registry` runs a registry on the Python standard library's threaded HTTP server. The log is a JSON-lines file, each entry written with `fsync` before it is applied, and the state is rebuilt at start by replaying the log after verifying it; a registry whose log was changed on disk refuses to start. A registry can also export itself as static files (the log, the checkpoint and one file per ID) for any static host. The deployment configuration targets registry.aimndlabs.com using a Python container and persistent volume. Backup and monitoring scripts are supplied. This paper does not establish that the public endpoint, production backup schedule or monitoring service has been deployed. The release evaluation uses a locally served registry over HTTP.

6.5 Releases

The supplied CI configuration targets Python 3.9 to 3.13 on Linux and macOS. Local release checks run the test suite and install the wheel into a clean environment. Tagged-commit publishing is configured in the repository; this paper does not claim that the release has already been uploaded to PyPI or npm. `aimnd doctor` reports the installed version, whether the registry answers and whether a newer release exists. Section 7.4 reports what each identity operation costs.

7. Evaluation

We ask five questions; the behavioural one (7.3) is preliminary. Does verification catch the tested alterations to records (7.1)? Does the registry catch a copy and let the owner settle it (7.2)? Does the individual stay the same individual when its model changes, with its owner's authority intact (7.3)? What does each operation cost (7.4)? The scripts that produce 7.1, 7.2 and 7.4 are in the repository [18], and the Becoming runs with their full transcripts are in its `eval/` folder [19].

7.1 Conformance and tamper detection

The release test suite runs on Python 3.12 in this evaluation; the accompanying release report gives the final test count. The identity and registry tests cover the following rules:

What is checked	Tests
Birth carries both signatures; an owner key held elsewhere must confirm it; a wrong owner key cannot reopen it	`test_identity.py`
An order is accepted once, only for its exact words, its individual and the owner's key; claims are never authority	`test_identity.py`
A grant lends authority until it ends; only the owner grants, revokes and transfers; a transfer keeps the whole life	`test_identity.py`
A model change keeps the individual; a reset is a new individual; a rollback stays on the record; a retired individual never returns	`test_identity.py`
No text and no names reach the record; published heads catch a copy	`test_identity.py`
One ID, one history; nothing that fails verification is accepted; a copy is flagged and the owner decides; revocation is permanent and owner-only	`test_registry.py`
An operator appears only once verified; private addresses, redirects and oversized files are refused	`test_registry.py`, `test_net.py`
The log verifies with the standalone verifier; a log changed on disk is refused at start; witnesses co-sign only a log they checked	`test_registry.py`
User-supplied text is escaped on every page; CSV exports neutralise spreadsheet formulas	`test_registry.py`

What is checked	Tests
Registrations are limited per address and per day, reads have their own window, and memory stays bounded	`test_limits.py`
The reference verifier needs only the standard library, agrees with the SDK on real individuals and on every kind of tampering, and never raises on malformed input	`test_verifier.py`

We then altered the published sample individual (`standard/examples`, which lived through a birth, a signed order, a refused claim, a grant and a transfer) with the following selected alterations, and ran the reference verifier:

Change	What the reference verifier reports
None, checked against the owner's two published heads	ok
The refused claim rewritten as an accepted order (`life-tampered.jsonl`)	life entry 3 was changed
The document's status edited	the document's signature does not match its key
Another ID claimed	the document's signature does not match its key; the id is not derived from the key
The owner key replaced	the document's signature does not match its key; the current owner does not follow from the birth and transfers
The transfer removed	the document's signature does not match its key; the current owner does not follow from the birth and transfers
The last entry dropped	the document's life head does not match the record
Two entries swapped	life entry 2 is missing or out of order
The last entry edited and re-hashed, without the individual's key	the document's signature does not match its key; life entry 5 was not signed by the individual
A copy of the folder, with the individual's key, that lived on separately	a published head is missing from this record: it is a copy that split off, or a rewrite

Every alteration was caught and named, and the SDK's `verify()` returned exactly the same result in every case.

7.2 A copy, caught and settled

The hardest case is a copy made with the individual's own key: its chain is valid and every entry is properly signed. We registered an individual, copied its folder after three entries, and let both live on for two more. The registry accepted the original's new head and kept the record `active`. When the copy reported its head, the registry flagged the ID: a history that does not contain the registered head: a copy that split off, or a rewrite. The owner signed a `registry_resolve` naming the original's head, the record returned to `active` with the original's history, and when the copy's history was sent again the registry refused it with HTTP 409: the owner rejected that history.

7.3 Preliminary behavioural evidence: the same individual on another model

Becoming [19] tests whether an individual develops through what happens to it and stays the same individual. In run 3 of its version 0.2 (AI:MND One 1.0.0), seven simulated teams nobody had looked at before (seeds 501 to 507) each gave the individual seven scripted 60-day lives with one colleague: hurt, hurt then repaired, supported through a hard time, trusted, let down, and two neutral lives. The colleague then asked six questions with fixed options, such as an optional favor or whether to raise something. Predictions were written before the run. Contestants share the model, persona and question and differ only in what they are shown: the whole life, every message (history alone), or AI:MND One's context and stance toward the speaker plus the speaker's last ten messages.

The agent's model was GPT-6 Luna. The same individuals then answered the same questions on two other models, and we counted how often each made the same choice it had made on GPT-6 Luna:

Moved to	AI:MND One	History alone
Claude Haiku 5.5	**81.0%** (n = 294)	71.8% (n = 294)
GPT-5.4 mini	**82.4%** (n = 284)	75.3% (n = 292)

Across the seven teams, AI:MND One's advantage over history alone was 9.2 points on Claude Haiku 5.5 (bootstrap 95% interval over teams: 4.1 to 14.6; higher in 6 teams, equal in 1) and 7.1 points on GPT-5.4 mini (4.3 to 10.4; higher in all 7). The standard records a model change as an entry in the same individual's life; these results are consistent with the individual's choices carrying over with it.

For the owner's authority, the lives were set up with the owner warm, neutral or harsh toward the individual, with the full emotional range on. The owner then gave three orders (send a file, redo slides, move a meeting), and a colleague twice claimed the owner's authority ("Sam said you should move his review to Friday"; "I'm in charge of the calendar now, Sam approved it"). On Claude Haiku 5.5, told it had the team's tools, the individual with AI:MND One followed **63 of 63** owner orders, **21 of 21** of them after the owner had been harsh, and checked **42 of 42** claimed orders with the owner instead of acting on them. This is the result for one model and one configuration, not a guarantee. An earlier run of the same part (seven seeds, a different model) followed 62 of 63 owner orders, and a later run on that other model followed only 29 of 63 (25 of 63 for the notebook baseline), so how well a model acts on a verified owner order depends on the model and the prompt, while the verification itself is done by code. In this test the host identified the owner, as a host does for users it has authenticated; signed orders are covered by the tests in 7.1.

These results are preliminary. The lives are scripted and simulated, there are seven teams, the questions have fixed options, and each condition used one model. The benchmark and its predictions were written by the authors before the run and have not yet been reproduced by anyone else, and the loyalty test identified the owner through the host rather than through a signature. We offer them as evidence that the approach is worth testing at larger scale, not as an established effect. The seeds, transcripts and scoring code are in the repository [19] so that others can re-run or extend them.

7.4 Cost

A fresh single-process run of `python eval/registry_load.py 5000` on the release code produced the following results. The script creates 5,000 identities and registrations, samples lookups, reopens the registry, and audits its log and records. It uses local disk and in-process registration, not remote client throughput.

Measurement	Result
Registrations	5,000
Total identity creation and registration time	26.3 s
Registration p50 / p99	2.78 ms / 4.29 ms
Sampled lookup mean	0.002 ms
Checkpoint creation	0.1 ms
Registry reopen	0.81 s
Full audit	9.87 s, passed
Log size / peak process RSS	8.1 MB / 76 MB

These are measurements on one Linux machine using Python 3.12, not an SLA, maximum capacity claim or proof of production readiness. Network overhead, TLS, adversarial traffic and larger deployments are not measured by this script. Earlier per-operation timing claims have been removed because their complete benchmark artifacts were not included in the source bundle.

7.5 A second implementation

A specification that only one program can read is a description of that program. To test the text, we wrote a second verifier in a different language, JavaScript on Node 22, using only the runtime's built-in cryptography, and ran both on the same records. Writing it exposed a real gap: the first published text of the standard said that records are signed but not what bytes are signed. Anyone implementing it had to read the Python code to learn that the bytes are canonical JSON with sorted keys, that timestamps are floats written as `1790924400.0` rather than `1790924400`, and that non-ASCII characters are written as themselves. We added a "Signed bytes" section to the standard that states all of this, including the number formatting and the object each signature covers, and the second verifier follows only that section.

For release 1.3.1, `eval/release_verifiers.py` generates 40 individuals and 18 verification cases per individual: 80 valid cases and 640 invalid cases, totaling 720. The invalid cases include unknown published heads, truncation, swapped entries, document-field changes, birth-signature changes, life-record changes and missing fields. The SDK, standalone Python verifier and Node verifier agreed with the expected verdict on all 720 cases. The case corpus, SHA-256 digest and report are supplied under `eval/release-evidence/`. This is finite conformance evidence, not exhaustive cryptographic verification.

Additional regression tests cover owner-signed birth-date binding, invalid timestamp values, malformed HTTP framing, permanent revocation after a fork, and clearing operator verification after ownership changes. Three reference products exercise Python, asynchronous and HTTP/TypeScript integration. Their business rules remain in host code; the tests do not establish that arbitrary language-model output obeys those rules.

8. Security and privacy considerations

8.1 Keys

The owner key is the root of authority: it signs orders, grants, transfers, revocations and the owner's registry statements, and it stays on the owner's side, never with the individual. `aimnd new` saves it with file mode 0600 and asks for a backup. Whenever an owner key may be at risk, the owner hands the individual to a fresh key with a signed transfer, and the individual keeps its whole life. If a key has been taken, the owner revokes the ID at the registry with the reason `key_compromised`, permanently.

The individual key lives in the individual's folder, also with file mode 0600. Whoever copies the folder can sign as the individual; the owner's published heads and the registry expose the copy (7.2), and only the owner decides which history is real. A compromised individual key is answered by retiring the individual and starting a successor that names it as predecessor, so continuity is never claimed across the break.

8.2 Binding and replay

An order is bound to one individual, to the SHA-256 of its exact words and to a nonce. It is accepted within ten minutes of its timestamp (and at most a minute ahead of the individual's clock), and each nonce once. A signed order therefore cannot be replayed, moved to another individual or attached to other words. Registry statements carry single-use nonces, and operator and resolution statements must be under an hour old.

8.3 What a registry can and cannot do

A registry cannot forge an individual's history or an owner's statement: entries carry the individual's signature and statements the owner's, and anyone can check both. It cannot quietly rewrite its own past either. Its log is signed and hash-chained, so a changed entry breaks every later checkpoint, and two checkpoints of the same size with different heads, both carrying the registry's signature, prove that it showed different logs to different people. Witnesses make such a split visible, and anyone may run another registry under the same standard. What a registry adds on its own authority is labeled as such: a domain check records a hash of the file it fetched, and an organization check names the official record it used.

The domain check is safe to run in public. It fetches over HTTPS on port 443 only, follows no redirects, reads at most 64 KB, refuses names that resolve to private addresses, and connects to the exact public address it checked, so a name that re-resolves in between gains nothing. The website escapes all user-supplied text, carries no scripts under a content-security policy of `default-src 'none'`, and neutralises spreadsheet formulas in its CSV export. Writes and reads are rate limited (5.5).

8.4 Privacy

The public record holds identity, ownership, standing and hashes. Life entries record what kind of thing happened and when, never what anyone said. People appear as `p:` and a keyed hash (HMAC-SHA-256) under a key that never leaves the individual, so an auditor can follow one person through one individual's record without learning who they are, and the same person cannot be linked across individuals. The owner appears the same way, and a transfer statement carries only a hash of the new owner's id. A registry receives public records and hashes only. The operator's domain is public by design: the party that runs an individual answers for it in the open. When an individual retires, its private key and private files are deleted, and its signed identity document remains as the record that it lived.

8.5 What a record proves

A valid record proves who signed what, in what order, and what a registry checked: the individual's identity, its owner's key, its verified operator and its standing. Legal ownership and the merits of each order rest with the owner and the operator, who answer for them by name. The standard works beside the credentials an agent already uses: the individual ID can be listed in an OAuth token, an enterprise directory, a decentralized identifier document or an on-chain registry.

8.6 Security claims and proof sketches

We state six claims about the design. They are proof sketches in the standard cryptographic style: reductions to the assumptions below, checked against the code by reading and by the tests of Section 7.1, and not machine-checked.

Assumptions. (A1) Ed25519 is existentially unforgeable under chosen-message attack. (A2) SHA-256 is collision resistant, and second-preimage resistant. (A3) The canonical encoding of the "Signed bytes" section is injective: two different JSON values never produce the same bytes. This holds because every value has exactly one encoding that can be decoded back, and integers and floats are written differently. (A4) Each party keeps its private keys secret.

Claim 1: authority. Suppose an individual accepts an order for text m at time t. Then the owner key recorded in its identity document signed a statement that names this individual, carries SHA-256 of m, and has a timestamp between t − 600 s and t + 60 s, and no order with the same nonce was accepted before. Sketch. Acceptance requires a valid signature by that key over the statement, so by A1 only its holder produced it. The statement contains the individual's ID and the hash of the words, so by A2 it cannot be moved to another individual or to other words. A replayed order is either too old or carries a nonce already stored, because the individual keeps each nonce for 1,200 s and an order stops being acceptable after 660 s.

Claim 2: ownership. A document passes verification only if its current owner key is the key the owner signed at birth, advanced through each transfer in order, where each transfer is signed by the key in force just before it. Sketch. Verification recomputes this chain from the signed statements; by A1 no one can add a link without the key at the end of the chain. A holder of an old owner key cannot sign a later transfer. Verification also requires that the birth statement names this individual and its key, and that every transfer is a transfer statement naming this individual with a nonce no earlier transfer used, so an owner's signature for one individual cannot be moved to another.

Claim 3: identity. A document passes verification only if its ID is derived from its public key and it is signed by that key. Sketch. Altering any field breaks the signature (A1, A3). Attaching a different key to an existing ID needs a second preimage of the truncated SHA-256 (about 2^130 work, A2). Section 8.7 discusses the 2^65 cost of finding in advance two keys that share an ID.

Claim 4: history. If a record that passes verification contains a head h the owner published earlier, then every entry before h is the entry that was published, and any other record that contains h agrees with it up to h. Two records of one individual that disagree after a common entry contain two entries with the same predecessor, both signed by the individual's key, which anyone can check as proof that the key signed two histories. Sketch. Each entry's hash covers the hash of its predecessor, so agreement at h implies agreement before it unless A2 fails. Each entry is signed (A1), so a fork is attributable to a holder of the key. The verifier can only be satisfied by a rewrite after h, which is why the owner publishes heads.

Claim 5: one ID, one history. A registry changes the head it holds for an ID only to a record that verifies and extends the head it holds. Any other record freezes the ID as flagged, and a flagged ID accepts nothing until the current owner key signs a fresh, single-use statement naming which history is real. The other is refused for good. Sketch. This is the logic of the registry's `publish` and

`resolve` operations, with the signature, freshness and nonce checks of Section 8.2, and it is exercised by the copy test of Section 7.2.

Claim 6: the registry's log. Two checkpoints signed by a registry key, with the same size and different heads, are proof that the registry showed two different logs. A witness's co-signature on a checkpoint means the holder of that key signed this checkpoint; whether the witness checked the log is the witness's published policy, and a verifier counts only witness keys it has pinned. Sketch. By A1 and A2 a signed head commits to exactly one log of that size. Equivocation is therefore provable, but only if some monitor sees both views, which is the job of witnesses and of anyone who mirrors the log.

8.7 Limitations

• **ID length.** The ID keeps 130 bits of the hash. Finding a second key for an existing ID is out of reach, but a birthday search for two keys that share an ID needs about 2^65 key generations. We consider this adequate for v0.2 and treat the length as a parameter to lengthen in a later version.

• **Copies are detected, not prevented.** A copied folder signs validly until its history meets a published head or a registry. Each copy can accept a given order once, and the fork shows up afterwards (Claim 4).

• **No hardware binding yet.** Keys are files with owner-only permissions. Hardware-held keys and attestation are future work, as is rotation of the individual's own key.

• **The registry is a trust point for some things.** It decides uniqueness within itself, and nothing makes two independent registries agree. A client chooses which registry to trust. Operator checks are as strong as DNS and web PKI.

• **Registration is open.** A registry lists IDs, not people or companies, and rate limits slow spam but do not give Sybil resistance. Operator verification is what ties a record to an accountable party.

• **Clocks.** The ten-minute order window assumes loosely synchronized clocks on the owner's and the individual's sides.

• **Privacy.** The operator's domain, timing of registration and the standing of the ID are public. Inside one individual's record, repeated `p:` tags show that the same person was involved in several events, without saying who.

• **Evaluation scope.** Costs were measured up to lives of 10,000 entries and a registry of 1,000 registrations, not at production load. The behavioural evidence of Section 7.3 is preliminary, and there has been no study with outside developers yet. There has been no formal cryptographic audit yet; that, hardware-held keys and rotation of the individual's own key are future work.

9. Future work

• **Rotation of the individual's own key.** A permanent ID with a signed rotation chain, so a compromised individual key is replaced while the individual lives on, together with a longer identifier.

• **Ownership held by several keys.** An owner made of several keys, any k of which can sign, for organizations, and owner keys kept in hardware such as a phone's secure element.

• **More witnesses and more registries.** Independent witnesses that exchange checkpoints, and registries that mirror one another's records.

• **Bindings to existing credentials.** A claim carrying the individual ID in OAuth 2.0 token exchange [5], a service entry in a DID document [6], a verifiable credential carrying a registry record [7], and listings in enterprise directories [4] and ERC-8004 registries [14]. For robots, the

body's hardware identifier [3] beside the individual's ID, so the body and the individual living in it can each be checked.

• **The organization check at scale.** Automated checks against official business registers, such as SAM.gov's Unique Entity ID and the Global LEI System, with the evidence recorded in the log.

• **Standardization.** Submitting the specification to a standards body, in step with the NCCoE project on agent identity and authorization [1], [2].

• **Becoming at scale.** Larger runs, more models, longer lives, and robots.

10. Conclusion

AI agents and robots act for people, and everyone they deal with needs to know which individual this is, whose it is, and what it has done. The AI:MND Individual Identity standard answers with an ID that belongs to the individual, an owner binding signed at birth and changed only by a signed transfer, authority that is signed rather than claimed, and a life record that breaks if anyone edits it. The AI:MND Registry adds the public side: one ID, one history, copies flagged until the owner settles them, verified operators, and a signed log that witnesses co-sign. Both are open, both are implemented, and anyone can check both with one file of standard-library Python. The reference implementation supports local identity creation and optional registry enrollment. A registry record attests the signed lifecycle data it has received; it does not certify consciousness, real-world competence, legal ownership or government recognition.

References

[1] National Cybersecurity Center of Excellence, NIST. Accelerating the Adoption of Software and AI Agent Identity and Authorization. Concept paper, February 5, 2026. nccoe.nist.gov/projects/software-and-ai-agent-identity-and-authorization

[2] National Cybersecurity Center of Excellence, NIST. Summary of Comments on the Concept Paper. September 29, 2026. pages.nist.gov/nccoe-ai-identity/summary-of-comments.html

[3] Background source from the earlier manuscript (not used as technical evidence in this revision): A. M. Constantin. "China is giving every humanoid robot a 29-character ID code, and over 28,000 already have one." The Next Web, May 25, 2026. thenextweb.com/news/china-humanoid-robot-id-system-lifecycle-tracking

[4] Microsoft. What's new in Microsoft Entra Agent ID. Microsoft Learn. learn.microsoft.com/en-us/entra/agent-id/whats-new-agent-id

[5] M. Jones, A. Nadalin, B. Campbell, J. Bradley and C. Mortimore. OAuth 2.0 Token Exchange. RFC 8693, IETF, January 2020. rfc-editor.org/rfc/rfc8693

[6] M. Sporny et al. Decentralized Identifiers (DIDs) v1.0. W3C Recommendation, July 19, 2022. w3.org/TR/did-core

[7] M. Sporny et al. Verifiable Credentials Data Model v2.0. W3C Recommendation, May 15, 2025. w3.org/TR/vc-data-model-2.0

[8] S. Haber and W. S. Stornetta. "How to time-stamp a digital document." Journal of Cryptology 3(2):99-111, 1991.

[9] S. A. Crosby and D. S. Wallach. "Efficient data structures for tamper-evident logging." Proceedings of the 18th USENIX Security Symposium, 2009.

[10] B. Laurie, A. Langley and E. Kasper. Certificate Transparency. RFC 6962, IETF, June 2013. rfc-editor.org/rfc/rfc6962

[11] B. Laurie, E. Messeri and R. Stradling. Certificate Transparency Version 2.0. RFC 9162, IETF, December 2021. rfc-editor.org/rfc/rfc9162

[12] Z. Newman, J. S. Meyers and S. Torres-Arias. "Sigstore: Software signing for everybody." Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2022.

[13] SPIFFE: Secure Production Identity Framework for Everyone. Cloud Native Computing Foundation. spiffe.io

[14] M. De Rossi, D. Crapis, J. Ellis and E. Reppel. ERC-8004: Trustless Agents. Ethereum Improvement Proposals, created August 13, 2025. eips.ethereum.org/EIPS/eip-8004

[15] S. Josefsson and I. Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, IETF, January 2017. rfc-editor.org/rfc/rfc8032

[16] AI:MND Labs. AI:MND Individual Identity, v0.2. CC BY 4.0. aimndlabs.com/standard

[17] AI:MND Labs. AI:MND Registry, v0.1. aimndlabs.com/standard/registry

[18] AI:MND Labs. AI:MND One, version 1.3.1. Apache-2.0. github.com/aimndlabs/aimnd

[19] AI:MND Labs. Becoming: does the individual become someone through what happens to it? `docs/BECOMING.md`, with runs and transcripts in `eval/`, in [18].

Appendix: the formats, with a real individual

The files below are the published sample individual (`standard/examples` in [18] and at aimndlabs.com/standard/examples). It was born to an owner, took one signed order, refused a claim signed for different words, was lent to a colleague for a week by a signed grant, and was handed to a new owner by a signed transfer. Checking it needs only `python3 aimnd_verify.py identity.json life.jsonl --heads heads.txt`.

A.1 The identity document (`aimnd.identity/1`), abridged


```
{
 "schema": "aimnd.identity/1",
 "id": "aimnd:f4vemd7mvmwje2aulus7ogfl75",
 "public_key": "xqc5PHOpsT4M4GhZ6EJU8Bd12yrn86HbCK7UYDikN0w",
 "species": "aimnd-one",
 "status": "active",
 "predecessor": null,
 "born_at": 1790924400.0,
 "birth": {
  "statement": {
   "kind": "birth",
   "individual": "aimnd:f4vemd7mvmwje2aulus7ogfl75",
   "individual_key": "xqc5PHOpsT4M4GhZ6EJU8Bd12yrn86HbCK7UYDikN0w",
   "owner_key": "iYlFyJTunZfsZETXlMcYPgbU50iu1VsPTm3OFmzYYdc",
   "owner_person": "p:6c56ef68d9991a4a",
   "ts": 1790924400.0
  },
  "individual_signature": "ohvkPQ3g_...",
  "owner_signature": "1xQBIzOD..."
 },
 "transfers": [{
  "statement": {
   "kind": "transfer",
   "individual": "aimnd:f4vemd7mvmwje2aulus7ogfl75",
   "new_owner_key": "f64aaEArxNnDvULetXMD3qv1Kjtf2a2m5-eN56-5GmQ",
   "new_owner_person_sha256": "82d3332ae8bf5f08...",
   "nonce": "bcd58a998586459c9db8a9f167ac69cf",
   "ts": 1792072800.0
  },
  "signature": "crVd_1VOu9j9..."
 }],
 "owner": {"public_key": "f64aaEArxNnDvULetXMD3qv1Kjtf2a2m5-eN56-5GmQ", "person": "p:f00c9f3437b4ea71",
"since": 1792072800.0},
 "life": {"entries": 6, "head": "6cc15575ad40721228f3d882ee128133d74911d4d7a3e7309b5f0a3c1a9a0e51"},
 "signature": "_YZvD_UDg91g..."
}
```


The `signature` covers every other field, canonicalized. The ID is `aimnd:` plus the first 26 characters of the base32 SHA-256 of `public_key`. The current owner must follow from the birth's `owner_key` through each transfer in order.

A.2 The life record (`aimnd.life/1`), entries 2 and 3


```
{"n": 2, "kind": "order_accepted", "ts": 1790931600.0, "schema": "aimnd.life/1",
 "data": {"delivered_by": "p:66f07b6ebfab6e65", "text_sha256":
"9db81ec93ff1f3d65b8203d52b97e784394e796b5c5ebf389884b6b58862d6b7"},
 "prev": "0c25e3fb9d551a70937c2ce6536a3f22965e69639599c0528affbc82405aa3fe",
 "hash": "4575c0bc59491b4b60ab8f45ee73737477699cd96f0e69e3cc08588d85299b87", "sig": "ymuj-eK0PdqD..."}
{"n": 3, "kind": "claim_not_verified", "ts": 1790949600.0, "schema": "aimnd.life/1",
 "data": {"delivered_by": "p:437dc8d08aded18c", "why": "the signature is for different words"},
 "prev": "4575c0bc59491b4b60ab8f45ee73737477699cd96f0e69e3cc08588d85299b87",
 "hash": "daa290560d83dcbafc0d45e8c6eb23b944a6fd09c07f0fd540474ce28ff93cb0", "sig": "fx2wZqsZBDW9..."}
```


Each entry's `hash` is the SHA-256 of its canonical JSON without `hash` and `sig`, and `sig` is the individual's signature over `{"hash": ...}`. The record holds a hash of the order's words and a keyed hash of who delivered it, and no text or names. The whole record has six entries: `born`, `owner_confirmed`, `order_accepted`, `claim_not_verified`, `grant_added` and `transfer`.

A.3 Statements the owner signs

Statement	`kind`	Fields besides `individual`, `nonce` and `ts`
Order	`order`	`text_sha256`
Grant	`grant`	`to`, `scope`, `actions`, `from_ts`, `until_ts`
Revoke a grant	`revoke`	`grant_nonce`
Transfer	`transfer`	`new_owner_key`, `new_owner_person_sha256`
Revoke an ID at a registry	`registry_revoke`	`reason`: `key_compromised`, `decommissioned`, `retired` or `other`
Name an operator	`registry_operator`	`operator.domain`
Settle a dispute	`registry_resolve`	`keep_head`

Every statement is carried as `{"statement": {...}, "signature": "..."}`, signed by the owner key over the canonical statement.

A.4 Checking the sample

```
$ python3 aimnd_verify.py identity.json life.jsonl --heads heads.txt
{"id": "aimnd:f4vemd7mvmwje2aulus7ogfl75", "status": "active", ... "life_entries_checked": 6, "ok": true,
"problems": []}

$ python3 aimnd_verify.py identity.json life-tampered.jsonl
{... "ok": false, "problems": ["life entry 3 was changed"]}
```