Independent Submission
Internet-Draft
Intended status: Informational
Expires: 28 November 2026

R. Jackson

SAIHM (Sovereign AI Horizontal Memory)
27 May 2026

The Sovereign AI Horizontal Memory (SAIHM) Protocol
draft-saihm-memory-protocol-01

Abstract

This document defines the Sovereign AI Horizontal Memory (SAIHM) protocol, a memory layer for AI agents that supports post-quantum identity binding, public-chain audit anchoring, per-cell encryption with wallet-derived keys, revocable sharing contracts, and cryptographic right-to-erasure aligned with Article 17 of EU Regulation 2016/679 (GDPR).

SAIHM is the memory-layer protocol companion to the Model Context Protocol (MCP). MCP standardizes how AI agents reach tools and contextual data sources; SAIHM standardizes how AI agents persist, share, and erase memory across sessions, models, and vendors.

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 28 November 2026.

Copyright Notice

Table of Contents

1. Introduction

This document is published through the Independent Submission Stream of the RFC Series, as defined in [RFC4846] and described in [RFC8730]. It is not an Internet Standard, and it has not been reviewed for, nor does it necessarily reflect, the rough consensus of the IETF community.

1.1. Motivation

The Model Context Protocol [MCP], donated to the Agentic AI Foundation [AAIF] under the Linux Foundation on 9 December 2025, defines how an AI agent reaches external tools and contextual data sources. MCP solves the tool-access layer of the agent stack. It does not standardize how an agent persists memory across sessions, models, or vendors. Here, "memory" refers to the holder's content-addressable cells (Section 2.1), identified by their content (cellId), not the bytes of an addressable memory in the computer-architecture sense (Section 1.3).

Production AI agents today rely on one of several non-portable approaches: a local file system, a vendor-specific session log, or a vector database. None of these provides post-quantum identity binding, public-chain audit, cryptographic erasure aligned with Article 17 of GDPR [GDPR], or wallet-bound sovereignty that prevents an operator from reading cell content.

This document specifies the Sovereign AI Horizontal Memory (SAIHM) protocol. SAIHM is a memory-layer protocol that an MCP-capable agent may attach to gain durable, sovereign, revocably-shareable, cryptographically-erasable memory. The protocol is operator-agnostic, vendor-agnostic, and chain-agnostic. Where a public-chain audit anchor is named, it is named as a reference-deployment property, not as a protocol mandate.

1.2. Scope

This document defines:

* The cell shape (encrypted memory unit with canonical metadata).

* Identity binding using a post-quantum digital signature algorithm (ML-DSA-65, [FIPS204]).

* An encryption envelope deriving a per-cell DEK from the holder's wallet through a canonical HKDF chain [RFC5869].

* An audit anchor profile, with a reference deployment on a public chain.

* Sharing contracts (temporary, permanent, syndicate).

* Cryptographic erasure semantics (DEK destruction + tombstone + content-address blacklist) aligned with GDPR Article 17.

* An MCP binding consisting of eight canonical tools.

This document does NOT define:

* A specific blockchain (the protocol is chain-agnostic).

* Vector-database semantics (orthogonal).

* Agent runtime semantics (covered by MCP and runtime implementations).

1.3. 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.

AEAD: Authenticated Encryption with Associated Data, as defined in [RFC5116]. SAIHM uses an AEAD cipher to encrypt cell payloads; AES-256-GCM ([NIST-SP-800-38D]) is RECOMMENDED.

canonical request envelope: a CBOR map ([RFC8949]) carrying the inputs to a SAIHM tool call in a fixed key order. The envelope is the byte sequence over which the caller computes the ML-DSA-65 signature that authenticates the call. The fixed key order is enumerated in Section 4.

Cell: an encrypted memory unit, as defined in Section 2.1.

DEK: Data Encryption Key. A per-cell symmetric key derived via HKDF from the holder's identity key.

HKDF: HMAC-based Extract-and-Expand Key Derivation Function, as defined in [RFC5869].

Holder: the natural person, organization, or autonomous agent that owns the wallet seed bound to a SAIHM cell.

KEK: Key Encryption Key version identifier. A protocol-level counter that permits cryptographic agility through rotation without invalidating existing cells.

MCP: the Model Context Protocol [MCP]. In MCP, a "tool" is a function: a named operation with typed inputs and typed outputs exposed by an MCP host runtime to a model. SAIHM exposes its protocol entirely as eight such functions (Section 3).

memory: the aggregate set of SAIHM cells (Section 2.1) bound to a single holderId. SAIHM memory is content-addressable: each cell is identified by its cellId, and a holder’s memory is the union of all cells whose holderId matches the holder, minus those for which an erasure receipt has been anchored. This is distinct from "memory" in the computer-architecture sense (a contiguous addressable byte store).

memory-layer protocol: a network-protocol specification whose subject matter is the persistence, sharing, and erasure of memory belonging to a holder. A memory-layer protocol defines the encoding, authentication, integrity, and lifecycle of the at-rest data, without prescribing the application that produces or consumes it.

ML-DSA: Module-Lattice-Based Digital Signature Algorithm, as specified in [FIPS204]. SAIHM authenticates every operation with an ML-DSA-65 signature.

Operator: the entity providing storage, transport, and audit-anchor services to a holder. The operator MUST NOT have access to plaintext cell content.

Receipt: a signed record of a SAIHM operation, anchored on a public chain.

sentinel value: a designated in-band value that signals the absence of an ordinary value of the same type. In Section 2.5, expiresAt = 0 is the sentinel signalling "no time bound". The sentinel preserves a flat representation of the contract tuple; a separate boolean flag field would carry the same information at the cost of an additional field. This document retains the sentinel for representational economy.

storage tier: a named, operator-chosen persistence backend with

stated durability and retention properties. The tier value ("FILECOIN" is the example given in the reference deployment) is deployment metadata; the protocol does not prescribe a particular tier, and tier names have no protocol semantics beyond serving as a hint to the operator-side storage router.

UNIX epoch: the time reference defined in [IEEE-1003.1-2024] under "Seconds Since the Epoch". All SAIHM timestamps are encoded as unsigned integer seconds since the UNIX epoch.

wallet seed: a high-entropy secret value held by the holder, from which the holder's identity key is derived (Section 2.2). The wallet seed is generated, stored, and rotated by a wallet implementation external to this protocol; this document does not specify how the wallet seed is generated or stored.

1.4. Substrate prerequisites

A conformant SAIHM implementation depends on the following external substrates. Where a substrate is itself an IETF or NIST normative reference, it appears in the Normative References. Where a substrate is selected per deployment, the deployment is responsible for stating which selection it uses.

(a) A wallet implementation capable of producing a high-entropy secret of at least 256 bits, which the holder retains as the wallet seed (Section 2.2). The protocol does not specify how the wallet seed is generated, stored, or rotated.

(b) An implementation of HKDF as defined in [RFC5869], used for identity-key derivation and per-cell DEK derivation.

(c) An implementation of ML-DSA-65 conformant with [FIPS204].

(d) An implementation of an AEAD cipher conformant with [RFC5116]. AES-256-GCM ([NIST-SP-800-38D]) is RECOMMENDED.

(e) An implementation of SHA-256 ([FIPS180-4]) for cellId derivation.

(f) A CBOR codec conformant with [RFC8949] for cell and receipt encoding.

(g) A Model Context Protocol (MCP) host runtime exposing the SAIHM tool functions over the MCP transport.

(h) A public chain offering transactional finality and public-record properties, used for receipt anchoring. Per-deployment selection; the reference deployment uses COTI V2 (chain ID 2632500). Implementations MAY choose any chain offering equivalent properties.

(i) A storage tier exposing PUT, GET, and DELETE semantics over an opaque ciphertext blob. Per-deployment selection; the reference deployment uses a Filecoin tier.

2. Architecture

2.1. Cell shape

A SAIHM cell is the tuple:

<cellId, holderId, kekVersion, tier, cellNonce, ciphertext, signature, timestamp>

cellId: 32-byte content-addressable identifier. cellId MUST be computed as

```
cellId = SHA-256( kekVersion_be32 ||
                  cellNonce       ||
                  ciphertext )
```

where:

kekVersion_be32 is kekVersion encoded as a 4-byte big-endian unsigned integer;

cellNonce is the 16-byte cellNonce defined in Section 2.3;

ciphertext is the AEAD output, including the authentication tag;

|| denotes octet concatenation.

The output is the raw 32-byte SHA-256 digest. For transport on the MCP tool surface, cellId is conveyed as a lowercase hexadecimal string of length 64.

Including kekVersion and cellNonce in the derivation ensures that the same plaintext encrypted under a different KEK generation or with a different per-cell nonce produces a distinct cellId, eliminating cross-KEK and cross-nonce collisions.

At read time, implementations MUST recompute cellId from the cell's kekVersion, cellNonce, and ciphertext fields and MUST reject any cell whose stored cellId does not match the recomputation. Because cellNonce is a public field, this recomputation is the integrity check that prevents an operator from substituting cellNonce after the holder's signature is computed.

holderId: 32-byte holder identity, derived from the holder's wallet seed (Section 2.2).

kekVersion: 32-bit unsigned integer identifying the KEK generation in force at write time.

tier: short ASCII string naming the operator's storage tier (e.g., "FILECOIN").

cellNonce: 16-byte per-cell random value, used both in DEK derivation (Section 2.3) and in cellId derivation (above). cellNonce is a public field, transmitted with the cell to permit the cellId recomputation check above.

ciphertext: cell payload encrypted with the per-cell DEK under an AEAD cipher. AES-256-GCM is RECOMMENDED.

signature: ML-DSA-65 signature over the concatenation (cellId || holderId || kekVersion_be32 || timestamp_be64). When timestamp appears in any concatenated byte sequence over which a signature is computed, it MUST be encoded as 8 bytes in big-endian order (timestamp_be64).

timestamp: 64-bit unsigned integer. UTC seconds since the UNIX epoch as defined in [IEEE-1003.1-2024] ("Seconds Since the Epoch"). Implementations MUST NOT use a 32-bit signed representation. A 32-bit signed encoding of UNIX time overflows at 03:14:07Z on 19 January 2038; implementations encountering values produced by such an encoding MUST treat the affected receipts as malformed. The 64-bit unsigned representation supports values up to 2^64 - 1 seconds since the epoch, exceeding any practical time horizon by many orders of magnitude.

2.2. Identity binding (ML-DSA-65)

Holder identity is derived from a wallet seed via the canonical HKDF chain:

```
identityKey = HKDF(salt = "MPS-PQC-KEY-GEN-v1",
                   IKM  = walletSeed,
                   info = "MPS-AGENT-IDENTITY-v1",
                   L    = 64 bytes)
```

identityKey is then used as the seed for ML-DSA-65 keypair generation per [FIPS204]. The public component is holderId; the private component MUST NOT leave the holder's machine.

Every SAIHM operation MUST be authenticated by an ML-DSA-65 signature over the operation's canonical envelope. Verification MUST follow [FIPS204] using the public holderId.

2.3. Encryption envelope (HKDF chain)

For each cell, the holder derives a per-cell DEK:

```
DEK = HKDF(salt = KEK_v,
           IKM  = identityKey,
           info = cellNonce || "MPS-CELL-DEK-v1",
           L    = 32 bytes)
```

where KEK_v is the current KEK generation and cellNonce is a per-cell 16-byte random value. The DEK is used directly with the AEAD cipher to encrypt the cell payload.

KEK rotation is versioned. An operator MAY rotate the KEK under operator-defined policy; rotation does not invalidate existing cells because each cell carries its kekVersion. Implementations MUST verify the kekVersion at read time and reject cells whose KEK has been revoked.

2.4. Audit anchor (public chain)

For each operation (write, read, share, revoke, erase) the protocol emits a receipt:

<receiptId, cellId, operation, holderId, signature, timestamp>

The receipt is anchored on a public chain offering transactional finality and public-record properties. Anchoring MUST produce a chain-reachable identifier sufficient to reproduce the receipt independently.

Reference deployment: COTI V2 mainnet, chain ID 2632500, block explorer https://mainnet.cotiscan.io. Implementations MAY use any public chain offering equivalent finality and public-record properties. The chain choice is a deployment decision; this protocol does not mandate one.

2.5. Sharing contracts

A holder MAY share a cell with a grantee through a signed sharing contract:

<contractId, cellId, granteeId, mode, expiresAt, holderSignature>

mode is one of:

TEMPORARY: revocable; expiresAt MUST be no later than timestamp + 86400 seconds (24 hours).

PERMANENT: revocable; expiresAt MUST be the sentinel value 0 (no time bound).

SYNDICATE: A single sharing contract that names an ordered list of granteeIds rather than a single granteeId. The contract is atomic in two senses: all listed grantees gain access in a single signed event, and a single counter-signed contractRevocation removes access for every listed grantee simultaneously. expiresAt MAY be 0 (no time bound) or a future timestamp.

SYNDICATE validation by the operator:

(a) The operator MUST verify the holder’s ML-DSA-65 signature over the full contract, including the full grantee list.

(b) On saihm_recall by an entity X, the operator MUST include cells from any active SYNDICATE contract whose grantee list contains X’s holderId.

(c) On contractRevocation, the operator MUST cease serving the cell to every member of the grantee list from the moment the revocation is anchored on chain.

SYNDICATE use cases include sharing the memory of a multi-party deliberation (e.g., a deal syndicate or an incident-response team) with a fixed grantee set, where per-grantee TEMPORARY contracts would multiply administration cost and where issuing the same memory N times under N TEMPORARY contracts would multiply the anchored-receipt cost.

Revocation is performed via a counter-signed contractRevocation anchored on chain. Once anchored, the revocation takes effect immediately; subsequent recall attempts by the grantee MUST be rejected by the operator.

2.6. Cryptographic erasure

The erase operation is atomic at the cryptographic layer:

1. The holder destroys the DEK for the target cell.

2. A tombstone is emitted, carrying cellId and a timestamp.

3. The cell’s cellId is added to a public blacklist (see below).

4. An audit receipt is anchored on chain.

Because the operator never held the DEK (Section 2.3), the ciphertext at rest is computationally meaningless from the moment of DEK destruction. Erasure is irreversible by the operator.

The blacklist is a public, append-only set of cellIds. An entry is added to the blacklist when, and only when, the operator anchors an erasure receipt for that cellId on the audit chain. Each erasure receipt’s payload carries the cellId of the erased cell; the act of anchoring a valid erasure receipt is the act of adding the cellId to the blacklist.

The blacklist construct is mandatory: every successful saihm_forget operation MUST result in (i) destruction of the DEK by the holder, (ii) anchoring of an erasure receipt by the operator, and (iii) inclusion of the erased cellId in the blacklist by virtue of (ii).

The blacklist is reproducible from the audit chain by any observer: scan all anchored receipts whose operation field equals "FORGET" and project the cellId field. Operators MAY publish a cached, queryable form of the blacklist for latency-sensitive consumers; the cached form MUST be reproducible from the chain.

Implementations MUST consult the blacklist on every saihm_recall and MUST NOT return any cell whose cellId is blacklisted, regardless of whether the requesting party is the original holder, a TEMPORARY grantee, a PERMANENT grantee, or a SYNDICATE grantee. The protocol does not prescribe a particular index structure (e.g., Bloom filter, sorted set, on-chain Merkle accumulator); implementations choose one matching their scale and latency requirements.

3. Tool surface (MCP binding)

In the Model Context Protocol [MCP], a "tool" is a function: a named operation with typed inputs and typed outputs that an MCP host runtime exposes to the model. The eight SAIHM tools defined below are functions in this sense; each subsection gives the tool's signature in TypeScript-like notation, followed by its pre-conditions, behavior, post-conditions, and error returns.

SAIHM exposes the protocol entirely through MCP using eight tools. Each tool MUST authenticate the caller with an ML-DSA-65 signature (Section 2.2) over the tool's canonical request envelope.

The tool surface is intentionally bounded at eight. The bound is a design constraint imposed by this specification, not an upper limit imposed by MCP. The motivation is that a small, fixed tool surface is easier for agent runtime authors and security reviewers to audit, version, and reason about than an open-ended tool list.

Protocol evolution within this bounded surface proceeds through three mechanisms:

(i) KEK rotation: an operational change in which the same eight tools continue to operate under a new KEK generation (Section 2.3). Existing cells remain readable so long as their kekVersion has not been revoked.

(ii) Governance vote: a protocol-level parameter change proposed via saihm_governance_propose (Section 3.7) and ratified via saihm_governance_vote (Section 3.8). The set of governance scopes is itself governance-mutable.

(iii) Parameter changes within existing tool signatures: new sharing-contract modes (Section 2.5), new governance scopes (Section 3.7), or new payload fields are introduced by extending the typed input of an existing tool, not by adding a new tool.

A future major revision of this protocol MAY add or remove tools. Within this revision, the surface is fixed at eight.

3.1. saihm_remember

Signature: saihm_remember(content: string) -> cellId

Pre-conditions: Caller is the holder. The holder's wallet seed is

locally available to the holder-side runtime. The caller has constructed a canonical request envelope and signed it with the holder's ML-DSA-65 identity key.

Behavior: The holder derives a per-cell DEK (Section 2.3), encrypts content under an AEAD cipher to produce ciphertext, computes cellId (Section 2.1), signs the canonical envelope, and submits the cell to the operator. The operator persists the ciphertext to the holder-named storage tier and anchors a write receipt on the audit chain (Section 2.4).

Post-conditions: A cell with the returned cellId is retrievable via saihm_recall. A write receipt for cellId is anchored on the audit chain.

Errors: "no_wallet_seed", "envelope_signature_invalid", "operator_storage_unavailable", "operator_chain_unavailable", "rate_limited".

3.2. saihm_recall

Signature: saihm_recall(query?: string) -> cells[]

Pre-conditions: Caller is the holder, or a grantee under an active sharing contract that names the caller's holderId.

Behavior: The operator returns ciphertexts for all cells whose holderId matches the caller, plus ciphertexts from active sharing contracts naming the caller (Section 2.5). Cells whose cellId is on the public blacklist (Section 2.6) are omitted. The caller decrypts client-side and applies the optional query filter after decryption. query MUST NOT be transmitted to the operator.

Post-conditions: The returned cell set reflects the operator's view at recall time. Blacklisted cells MUST NOT appear in the returned set.

Errors: None beyond transport faults.

3.3. saihm_forget

Signature: saihm_forget(cellId: string) -> tombstone

Pre-conditions: Caller is the holder of the named cell. The caller's wallet seed is locally available.

Behavior: The holder destroys the per-cell DEK for cellId. The

operator anchors an erasure receipt on the audit chain, emits a tombstone, and adds cellId to the public blacklist (Section 2.6). After saihm_forget returns success, the cell content MUST be computationally non-recoverable by any party, including the operator.

Post-conditions: cellId appears on the public blacklist. The DEK for cellId is destroyed; the ciphertext at rest is computationally meaningless. Subsequent saihm_recall calls MUST NOT return the cell.

Errors: "not_holder", "cell_not_found", "already_erased", "operator_chain_unavailable".

3.4. saihm_status

Signature: saihm_status() -> { prs, bfsi, bfsi_R, bfsi_M, bfsi_window_start_ts, shards, contracts, governance }

Pre-conditions: Caller is the holder.

Behavior: The operator computes the dashboard fields per the schema specified in this subsection (and CBOR-encoded in Section 4).

Post-conditions: The returned dashboard reflects operator-recorded state at the time of call. The bfsi inputs (bfsi_R, bfsi_M, bfsi_window_start_ts) are sufficient for the caller to reproduce the bfsi computation.

Errors: None beyond transport faults.

Return value (an object with the following fields):

prs: IEEE 754 binary64 in the closed interval [0.0, 1.0]. Process Reliability Score: the fraction of the operator’s expected tool-call returns delivered within the operator’s published SLA window, computed over a rolling 30-day window.

bfsi: IEEE 754 binary64 in the closed interval [0.0, 1.0]. Byzantine Fault Score Index: the fraction of audit-chain receipts that match a corresponding holder-side tool-call event, computed over the same 30-day rolling window. Formally,

bfsi = 1 - (M / R)

where R is the count of operator-anchored receipts on the audit chain attributed to the holder over the window, and M is the count of those receipts for which no corresponding tool-call event is

attested in the holder's local event log. When R = 0 (no operator receipts attributed to the holder in the window), bfsi is defined as 1.0 by convention; this signals absence of fault evidence rather than positive integrity, and the bfsi_R field (below) discloses the underlying R = 0 to the holder. bfsi is not opaque; the inputs (R, M, and the window boundary) MUST be exposed to the holder through the same saihm_status call under the fields bfsi_window_start_ts, bfsi_R, and bfsi_M, so that any holder or auditor can reproduce the computation.

A bfsi below the operator's published operator-integrity threshold (the reference deployment publishes 0.99 as the threshold) is a signal to the holder that operator integrity is degrading and is grounds for migration to another operator.

bfsi_window_start_ts: Unsigned integer, UNIX epoch seconds. The start of the rolling 30-day window over which bfsi (and prs) is computed.

bfsi_R: Unsigned integer. The count of operator-anchored receipts on the audit chain attributed to the holder over the window.

bfsi_M: Unsigned integer. The count of those receipts for which no corresponding tool-call event is attested in the holder's local event log.

shards: a CBOR map keyed by tier name, value = unsigned integer count of cells stored under that tier.

contracts: a CBOR array of objects, each with fields contractId (32-byte hex), mode (string enum: "TEMPORARY" | "PERMANENT" | "SYNDICATE"), granteeIds (array of 32-byte hex), expiresAt (unsigned integer, UNIX epoch seconds).

governance: a CBOR array of objects, each with fields propId (32-byte hex), scope (string), opens_ts (unsigned integer), closes_ts (unsigned integer), tally_for (unsigned integer), tally_against (unsigned integer), tally_abstain (unsigned integer).

The full CBOR schema for the saihm_status return appears in Section 4.

3.5. saihm_share

Signature: saihm_share(cellId: string, granteeAgentId: string | string[], mode: "TEMPORARY" | "PERMANENT" | "SYNDICATE") -> contractId

Pre-conditions: Caller is the holder of cellId.

Behavior: The holder emits a signed sharing contract granting access per the mode semantics (Section 2.5). For SYNDICATE mode, granteeAgentId is an array; for TEMPORARY and PERMANENT, granteeAgentId is a single value.

Post-conditions: The named grantee(s) gain saihm_recall access to cellId per the mode's lifecycle. A share receipt is anchored on the audit chain.

Errors: "not_holder", "cell_not_found", "invalid_mode", "syndicate_grantees_empty", "operator_chain_unavailable".

3.6. saihm_revoke_share

Signature: saihm_revoke_share(contractId: string) -> revocationId

Pre-conditions: Caller is the holder party to the contract named by contractId.

Behavior: The holder emits a counter-signed contractRevocation. The operator ceases serving cell access to the contract's grantee(s) from the moment the revocation is anchored on chain.

Post-conditions: The revocation is anchored on the audit chain. Subsequent saihm_recall by the grantee(s) MUST NOT return the cell under this contract.

Errors: "not_holder", "contract_not_found", "already_revoked", "operator_chain_unavailable".

3.7. saihm_governance_propose

Signature: saihm_governance_propose(scope: string, payload: object) -> propId

Pre-conditions: Caller satisfies the deployment's eligibility-to-propose predicate (Section 3.9). scope is one of the deployment's declared proposal scopes.

Behavior: The caller submits a signed proposal. The operator anchors the proposal receipt on the audit chain. The voting window opens.

Post-conditions: A proposal with propId is published; it accepts saihm_governance_vote calls for the deployment's voting window duration.

Errors: "ineligible_proposer", "unknown_scope", "operator_chain_unavailable".

3.8. saihm_governance_vote

Signature: saihm_governance_vote(propId: string, vote: "FOR" | "AGAINST" | "ABSTAIN") -> voteReceipt

Pre-conditions: Caller satisfies the deployment's eligibility-to-vote predicate (Section 3.9). propId names an open proposal.

Behavior: The caller submits a signed vote. The operator anchors the vote receipt on the audit chain. The proposal's tally updates.

Post-conditions: The vote is anchored. The proposal's tally on the audit chain reflects the vote per the deployment's vote-weight function. Vote tallies MUST be reproducible from the audit chain.

Errors: "ineligible_voter", "proposal_not_found", "proposal_closed", "operator_chain_unavailable".

3.9. Governance form

This protocol defines a governance hook (saihm_governance_propose, saihm_governance_vote) but does not mandate a single governance form. The set of governance scopes is itself governance-mutable, so a deployment is free to evolve its governance policy without revising this protocol.

Every deployment MUST publish its governance form, comprising at least the following parameters, at a stable URL referenced from the deployment's saihm_status output:

(a) Eligibility to propose: the predicate over a holderId that determines whether the holder may submit a proposal.

(b) Eligibility to vote: the predicate over a holderId that determines whether the holder may cast a vote.

(c) Vote weight: the function mapping a holderId to a non-negative real number, contributing to the tally.

(d) Threshold: the predicate over a tally that determines whether a proposal passes.

(e) Voting window: the duration in seconds during which saihm_governance_vote calls are accepted for a given propId.

(f) Proposal scopes: the enumeration of permitted scope values for saihm_governance_propose.

The reference governance form, used by the reference deployment, has the following values for the parameters above:

(a) Eligibility to propose: any holderId for which at least one saihm_remember receipt has been anchored on the audit chain.

(b) Eligibility to vote: same as (a).

(c) Vote weight: the holderId's balance of governance soul-bound tokens (SBTs) at the moment the proposal is anchored. SBTs are non-transferable, non-purchasable tokens minted to a holderId in exchange for verified, on-chain SAIHM protocol activity. The exact issuance schedule (which protocol events mint how many SBTs, at what cadence, and up to what cap) is deployment policy and is published at the deployment's governance URL.

(d) Threshold: (tally_for > tally_against) AND (tally_for + tally_against >= 0.10 * sum of all eligible SBT balances at proposal anchor time).

(e) Voting window: 1209600 seconds (14 days) from anchoring of the proposal receipt.

(f) Proposal scopes: enumerated at https://saihm.coti.global/governance.

Sybil-resistance note: the reference governance form above is hardened against Sybil attack by the soul-bound, non-transferable nature of the governance SBTs and by the requirement that SBTs be earned through demonstrated SAIHM protocol use rather than purchased or transferred. A party that spawns N wallets to multiply identity gains at most the minimum SBT yield per wallet (the floor that issuance policy permits for a newly active holderId); a long-standing legitimate holder accumulates SBTs over time at a rate that this Sybil strategy cannot match without performing equivalent legitimate protocol activity, by construction. Deployments that need a different Sybil-resistance posture (for example, identity attestation, KYC, or proof-of-stake at scale) MAY substitute an alternative governance form, subject to constraints (i)-(iii) below.

Implementations MAY substitute alternative governance forms (for example, stake-weighted, multi-signature board, or DAO-contract governance) so long as

(i) the saihm_governance_propose and saihm_governance_vote tool signatures remain unchanged;

(ii) vote outcomes are reproducible from the audit chain; and

(iii) the parameters (a)-(f) above are published at a stable URL referenced from saihm_status.

4. Wire formats

SAIHM uses Concise Binary Object Representation (CBOR) [RFC8949] for cell payloads and receipts, and JSON-RPC 2.0 over the MCP transport for tool calls. All timestamps appearing in CBOR-encoded payloads MUST be CBOR unsigned integers (CBOR major type 0); no SAIHM emitter or parser is permitted to clamp to a signed 32-bit window (Section 2.1).

A canonical cell payload (CBOR diagnostic notation):

```
{1: h'...cellId...',
 2: h'...holderId...',
 3: 1,
 4: "FILECOIN",
 5: h'...cellNonce-16-bytes...',
 6: h'...ciphertext...',
 7: h'...signature...',
 8: 1747526400}
```

The integer keys correspond to fields defined in Section 2.1: 1=cellId, 2=holderId, 3=kekVersion, 4=tier, 5=cellNonce, 6=ciphertext, 7=signature, 8=timestamp.

A canonical receipt (CBOR diagnostic notation):

```
{1: h'...receiptId...',
 2: h'...cellId...',
 3: "REMEMBER",
 4: h'...holderId...',
 5: h'...signature...',
 6: 1747526400}
```

For erasure receipts (operation = "FORGET"), the receipt payload's cellId field carries the cellId added to the blacklist (Section 2.6).

A canonical saihm_status return (CBOR diagnostic notation):

```
{"prs":                  0.997,
 "bfsi":                 1.0,
 "bfsi_window_start_ts": 1745020800,
 "bfsi_R":               0,
 "bfsi_M":               0,
 "shards":               {"FILECOIN": 42},
 "contracts":            [
   {"contractId":  h'...32-bytes...',
    "mode":        "TEMPORARY",
    "granteeIds":  [h'...32-bytes...'],
    "expiresAt":   1747612800}
 ],
 "governance":           [
   {"propId":         h'...32-bytes...',
    "scope":          "kek-rotation-cadence",
    "opens_ts":       1747000000,
    "closes_ts":      1748209600,
    "tally_for":      1240,
    "tally_against":  18,
    "tally_abstain":  5}
 ]}
```

5. Receipt and audit semantics

For each tool call that mutates state (saihm_remember, saihm_forget, saihm_share, saihm_revoke_share, saihm_governance_propose, saihm_governance_vote), the operator MUST emit a receipt anchored on the audit chain.

The receipt MUST include the operation type, the cellId (or contractId, or propId), the holderId, and the timestamp.

The receipt MUST NOT include cell plaintext, the DEK, or the wallet seed.

Receipts for erasure (saihm_forget) MUST also include the tombstone identifier and the cellId added to the blacklist (Section 2.6).

Read-only tool calls (saihm_recall, saihm_status) MAY emit receipts under operator policy; receipts for read-only calls MUST NOT include cell plaintext.

6. Security considerations

6.1. Post-quantum identity

ML-DSA-65 [FIPS204] is the protocol's authentication primitive. ML-DSA-65 is a NIST-selected post-quantum digital signature algorithm in the FIPS-204 family. An adversary with a cryptographically relevant quantum computer cannot forge SAIHM signatures.

Implementations MUST use the FIPS-204 parameter set named ML-DSA-65. Implementations SHOULD include the kekVersion in every signed envelope so that key-rotation events are themselves signed.

6.2. Sovereign key custody

The wallet seed and the derived identity key MUST NOT leave the holder's machine. The protocol does not provide key escrow. A holder who loses the wallet seed loses access to their cells; the operator cannot recover access on behalf of the holder.

This is intentional. The operator's inability to recover the wallet seed is the same property that prevents the operator from reading cell content.

6.3. Cryptographic erasure properties

Cryptographic erasure (Section 2.6) provides stronger evidence than logical (soft) deletion. Because the DEK is destroyed at the holder side, ciphertext at rest is computationally meaningless to any party, including the operator.

Backups and replicas store only ciphertext. DEK destruction propagates erasure semantics to every copy of the ciphertext, anywhere, automatically. An operator MUST NOT cache the DEK.

6.4. Operator threat model

The protocol assumes an honest-but-curious operator: the operator is expected to provide storage and transport honestly but MAY attempt to learn cell content.

Against an honest-but-curious operator:

* Cell plaintext is never accessible to the operator (encryption-before-egress + per-cell DEK).

* Cell signatures are reproducible from public keys, so the operator cannot inject forged cells.

* Receipt validity is verifiable against the public chain, so the operator cannot rewrite the audit trail.

Against a malicious operator (one that deviates from the protocol), additional measures (e.g., threshold-replicated storage, multi-chain receipt anchoring) may be advisable but are out of scope for this protocol layer.

7. Privacy considerations

7.1. GDPR Article 17 alignment

The saihm_forget operation provides cryptographic-grade evidence of erasure aligned with Article 17 of Regulation (EU) 2016/679 [GDPR]. The DEK is destroyed; a tombstone is published; the cellId is blacklisted; and a receipt is anchored on chain.

Article 17 requires erasure "without undue delay". saihm_forget is atomic at the cryptographic layer; there is no operator-side deletion queue that could fail to drain.

7.2. Minimization

The audit log records cellId, holderId, operation, and timestamp. It MUST NOT record cell plaintext, the DEK, or the wallet seed. An auditor observing the chain learns when and by whom each operation was performed, but not the content of any cell.

7.3. Audit log content

Receipts anchored on chain are public. Holders SHOULD treat the existence of a cell (cellId, holderId, timestamp) as publicly observable. Where the existence of a record is itself sensitive, holders SHOULD use additional countermeasures outside the scope of this protocol (e.g., pseudonymous holder identities, batched-operation timing).

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[FIPS180-4]
National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <https://csrc.nist.gov/pubs/fips/180-4/upd1/final>.

[FIPS204] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard", FIPS PUB 204, August 2024, <https://csrc.nist.gov/pubs/fips/204/final>.

[IEEE-1003.1-2024]
IEEE, "IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 8", IEEE Std 1003.1-2024, June 2024, <https://standards.ieee.org/ieee/1003.1/7700/>.

[NIST-SP-800-38D]
Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST SP 800-38D, November 2007, <https://csrc.nist.gov/pubs/sp/800/38/d/final>.

[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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

9.2. Informative References

[AAIF] Linux Foundation, "Agentic AI Foundation", 9 December 2025, <https://aaif.io/>.

[EU-AI-ACT] European Parliament and Council, "Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 (Artificial Intelligence Act)", June 2024, <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>.

[GDPR] European Parliament and Council, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation)", April 2016, <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.

[ISO-27001] ISO/IEC, "ISO/IEC 27001:2022 Information security management systems --- Requirements", October 2022.

[ISO-42001] ISO/IEC, "ISO/IEC 42001:2023 Information technology --- Artificial intelligence --- Management system", December 2023.

[MCP] Anthropic, "Model Context Protocol Specification", Donated to the Agentic AI Foundation under the Linux Foundation on 9 December 2025, 25 November 2025, <https://modelcontextprotocol.io/>.

[NIST-AI-RMF] National Institute of Standards and Technology, "AI Risk Management Framework 1.0", NIST AI 100-1, January 2023, <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf>.

[RFC4846] Klensin, J. and D. Thaler, "Independent Submissions to the RFC Editor", RFC 4846, DOI 10.17487/RFC4846, July 2007, <https://www.rfc-editor.org/info/rfc4846>.

[RFC8730] Brownlee, N. and R. Hinden, "Independent Submission Editor Model", RFC 8730, DOI 10.17487/RFC8730, February 2020, <https://www.rfc-editor.org/info/rfc8730>.

Appendix A. Reference deployment

The reference SAIHM deployment runs on the COTI V2 Helium mainnet (chain ID 2632500). Protocol implementations MAY anchor receipts on any public chain offering equivalent transactional finality and public-record properties. The chain choice is a deployment decision; the protocol does not mandate one.

The reference deployment publishes:

* npm reference implementation: @saihm/mcp-server (Apache 2.0).

* Block explorer for chain receipts: https://mainnet.cotiscan.io.

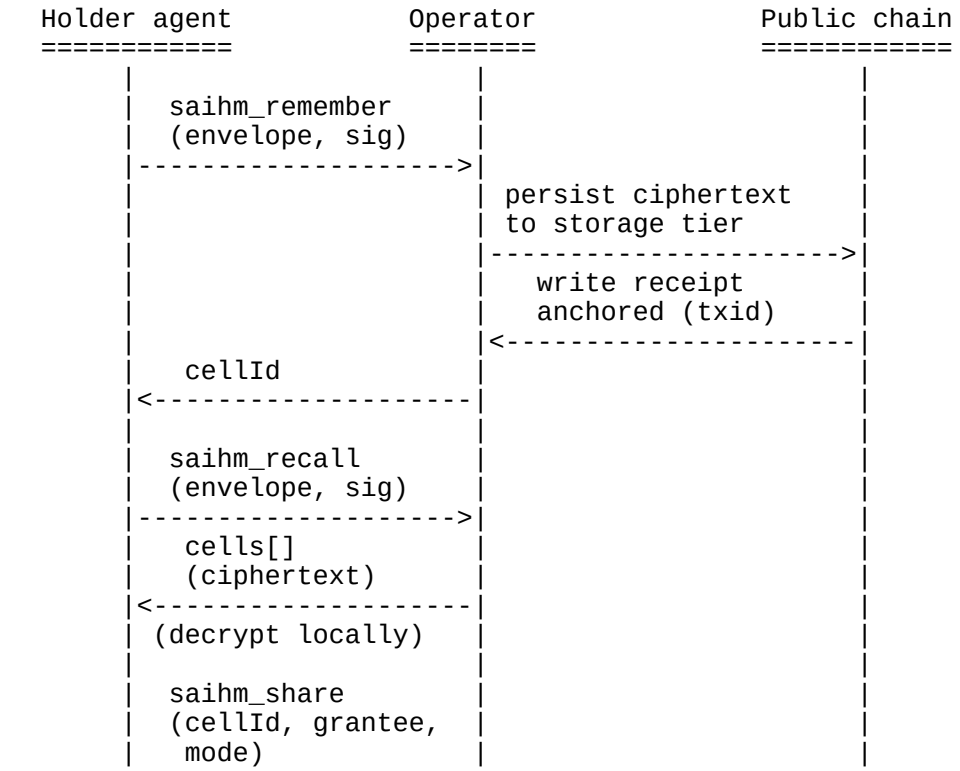
* Crosswalks to NIST AI RMF [NIST-AI-RMF], ISO/IEC 42001 [ISO-42001], ISO/IEC 27001 [ISO-27001], EU AI Act [EU-AI-ACT], GDPR Article 17 [GDPR], and MCP [MCP] at https://saihm.coti.global/standards.

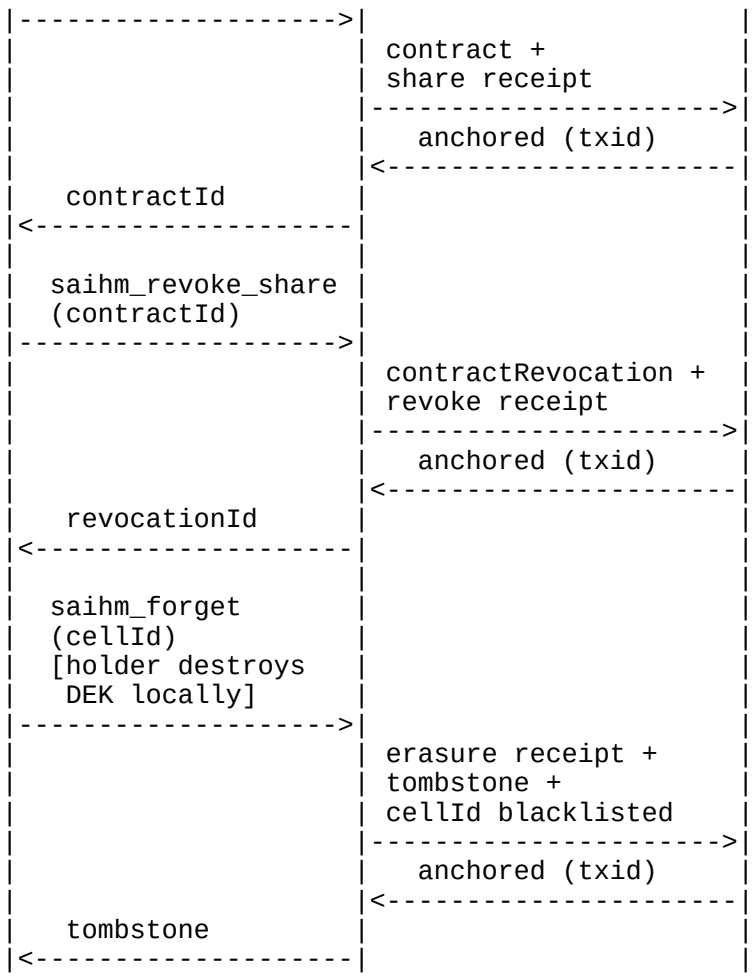
Appendix B. Worked example

This appendix walks through the canonical SAIHM cell lifecycle with concrete, illustrative values. All cryptographic inputs labelled "TEST" are derived from a well-known test wallet seed and MUST NOT be used for production cells.

B.1. Lifecycle sequence diagram

The canonical lifecycle saihm_remember -> saihm_recall -> saihm_share -> saihm_revoke_share -> saihm_forget, with the actor (holder agent, operator, public chain) and the message at each hop:





B.2. Cryptographic example values

The following values are derived from a labelled test wallet seed and are intended only to demonstrate the HKDF, AEAD, SHA-256, and CBOR derivations defined in the body of this document. Any FIPS-204-conformant ML-DSA-65 implementation, RFC 5869 HKDF implementation, and RFC 5116 AEAD implementation will reproduce these values from the inputs given.

Test wallet seed (32 bytes, hex), derived as SHA-256("SAIHM-TEST-VECTOR-001-DO-NOT-USE-IN-PRODUCTION"):

walletSeed (TEST) =
f068b8db8484d33bdbedd154bf5bf28e11fba330b79469e23595d6f738d7f5c6

HKDF identity derivation (Section 2.2):

```
identityKey =
  HKDF(salt = "MPS-PQC-KEY-GEN-v1",
       IKM  = walletSeed,
       info = "MPS-AGENT-IDENTITY-v1",
       L    = 64 bytes)
       =
  fdf786da8cb1f074393f9ad6dec1671e08085540e95c8463c9f2e15f437bbc5a
  2c267aeaf355e79d8968cc21846cdff7e16f498437de3d6b1fd290703eeed9c2
```

holderId (32 bytes, hex), presented as the test holder’s protocol-level identifier:

```
holderId (TEST) =
  ab4f746fd1520d2736854559d6751969ae9127f5dbc607d7298acbf1afb1f588
```

KEK generation: kekVersion = 1.

Per-cell nonce (16 bytes, hex):

```
cellNonce =
  25bd74b827789faacad8ffb7593c2359
```

Cell plaintext:

```
plaintext =
  "Hello, SAIHM. This is a test memory cell."
```

DEK derivation (Section 2.3):

```
DEK =
  HKDF(salt = KEK_v (4-byte big-endian of kekVersion=1),
       IKM  = identityKey,
       info = cellNonce || "MPS-CELL-DEK-v1",
       L    = 32 bytes)
       =
  a1e54f730c6c7a1048ae436c9d2b179821c6eddd7841f41e36215285f3ffd07a
```

AEAD encryption (AES-256-GCM, [NIST-SP-800-38D]) using the first 12 bytes of cellNonce as the GCM IV in this test vector; the protocol does not prescribe an AEAD-IV derivation, and deployments select one:

```
ciphertext (57 bytes, including 16-byte GCM tag) =
  595b0b9f77a8d95f29c5affa151bbcc18fdc3f7e5223792627919ee2d52f2bf2
  4933c35de4c1755559bee818ef54b48a388bcdd2e4a30fbde8
```

cellId derivation (Section 2.1):

```
cellId =
  SHA-256( kekVersion_be32 || cellNonce || ciphertext )
       =
  d851960a5b7754c5884c96bef5d615e666c8ad006e4ceebe028cd85aae8e7c2f
```

timestamp = 1747526400 (UTC 2025-05-18T00:00:00Z; encoded as 8 bytes big-endian when used in signature concatenation).

signature: ML-DSA-65 ([FIPS204]) over (cellId || holderId || kekVersion_be32 || timestamp_be64). The signature is 3309 bytes; the first 32 bytes are shown for illustration, and the full signature is reproducible from a FIPS-204-conformant ML-DSA-65 implementation given the test wallet seed above:

```
signature (first 32 of 3309 bytes) =
  dba1109b3d53e17290914cdb2e639c6cdad37aff8184537ba9b2c05c7fe8a994
  ... (3277 more bytes) ...
```

B.3. CBOR encoding walkthrough

The canonical CBOR encoding of the example cell, per the schema in Section 4:

```
{1: h'd851960a5b7754c5884c96bef5d615e666c8ad006e4ceebe028cd85aae8e7c2f',
 2: h'ab4f746fd1520d2736854559d6751969ae9127f5dbc607d7298acbf1afb1f588',
 3: 1,
 4: "FILECOIN",
 5: h'25bd74b827789faacad8ffb7593c2359',
 6: h'595b0b9f77a8d95f29c5affa151bbcc18fdc3f7e5223792627919ee2
      d52f2bf24933c35de4c1755559bee818ef54b48a388bcdd2e4a30fbde8',
 7: h'dba1109b3d53e17290914cdb2e639c6cdad37aff8184537ba9b2c05c
      7fe8a994 ... (3277 more bytes; 3309 total)',
 8: 1747526400}
```

Byte-level CBOR encoding (with one line of commentary per field; ML-DSA-65 signature bytes elided for brevity):

```
  A8                # map(8)
     01             #   unsigned(1)        // key=1 (cellId)
     58 20          #   bytes(32)
        d851960a5b7754c5884c96bef5d615e666c8ad006e4ceebe028cd85aae8e7c2f
     02             #   unsigned(2)        // key=2 (holderId)
     58 20          #   bytes(32)
        ab4f746fd1520d2736854559d6751969ae9127f5dbc607d7298acbf1afb1f588
     03             #   unsigned(3)        // key=3 (kekVersion)
     01             #   unsigned(1)        // value = 1
     04             #   unsigned(4)        // key=4 (tier)
     68             #   text(8)
        46494c45434f494e            // "FILECOIN"
     05             #   unsigned(5)        // key=5 (cellNonce)
     50             #   bytes(16)
        25bd74b827789faacad8ffb7593c2359
     06             #   unsigned(6)        // key=6 (ciphertext)
     58 39          #   bytes(57)
        595b0b9f77a8d95f29c5affa151bbcc18fdc3f7e5223792627919ee2d52f2bf2
        4933c35de4c1755559bee818ef54b48a388bcdd2e4a30fbde8
     07             #   unsigned(7)        // key=7 (signature)
     59 0CED        #   bytes(3309)
        dba1109b3d53e17290914cdb2e639c6cdad37aff8184537ba9b2c05c7fe8a994
        ... (3277 more signature bytes; elided for brevity) ...
     08             #   unsigned(8)        // key=8 (timestamp)
     1A 68292300    #   unsigned(1747526400)
```

The corresponding write receipt (operation = "REMEMBER"), per the schema in Section 4:

```
{1: h'(32-byte receiptId, anchored on chain at txid)',
 2: h'd851960a5b7754c5884c96bef5d615e666c8ad006e4ceebe028cd85aae8e7c2f',
 3: "REMEMBER",
 4: h'ab4f746fd1520d2736854559d6751969ae9127f5dbc607d7298acbf1afb1f588',
 5: h'(3309-byte ML-DSA-65 signature over receipt envelope)',
 6: 1747526400}
```

Author's Address

Russell Jackson
SAIHM (Sovereign AI Horizontal Memory)
Email: architect@saihm.coti.global
URI: https://saihm.coti.global/