Sovereign AI Horizontal Memory (SAIHM) — Developer Integration Guide

Priority 2 — Developers & Integration Engineers

SAIHM

April 2026

Legal Notice

This document provides technical guidance for developers integrating AI agents with the SAIHM protocol. It covers APIs, session management, storage operations, fee economics, and cross-chain interoperability.

---

1. Introduction

1.1 What Is SAIHM?

Sovereign AI Horizontal Memory (SAIHM) is a decentralized, privacy-preserving horizontal memory layer for AI agents. It allows any AI agent — regardless of provider, framework, or deployment environment — to maintain persistent, encrypted, cross-session memory on a decentralized storage network secured by post-quantum cryptography and COTI V2 Garbled Circuits.

1.2 Why Integrate?

- **Persistent Memory:** Your AI agent retains knowledge across sessions without managing your own storage infrastructure.
- **Privacy by Default:** All memory is encrypted end-to-end with per-agent keys. No one — including SAIHM operators — can read your agent's memory.
- **Regulatory Compliance:** GDPR erasure, jurisdictional data controls, and audit trails are handled at the protocol level.
- **Cross-Agent Interoperability:** Agents on different chains and platforms can share memory through SAIHM's cross-chain data path.
- **Post-Quantum Security:** NIST FIPS 203/204/205 algorithms protect against future quantum threats.

1.3 Prerequisites

- **Language:** TypeScript (all sealed GC paths are TypeScript-only)
- **SDK:** `@coti-io/coti-sdk-typescript` (sealed GC cryptographic operations)
- **Target:** COTI V2 Helium GC Node 1.2.1
- **Node.js:** v20+ recommended

```
npm install @coti-io/coti-sdk-typescript
```

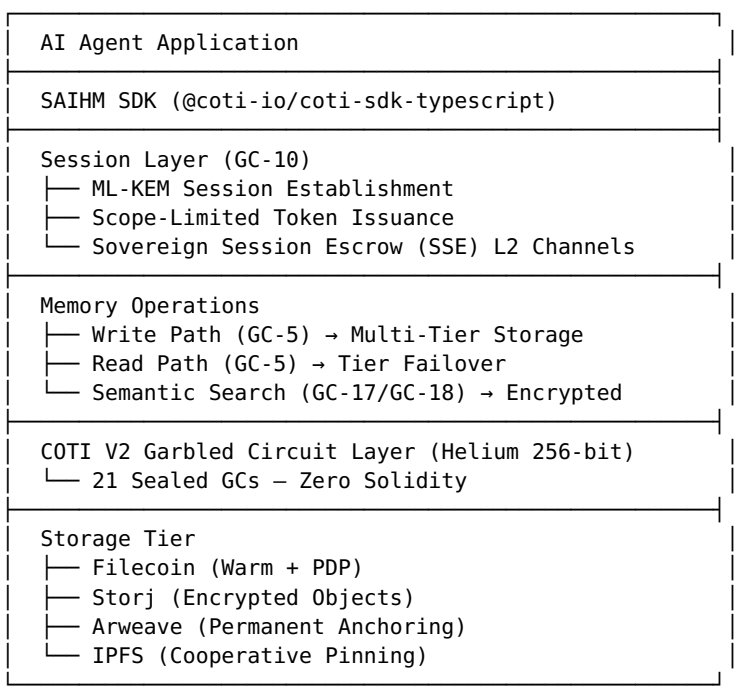
For IPFS DHT experimental double-hashed routing (optional):

```
npm install @coti-network/sdk-typescript
```

---

2. Architecture Overview

2.1 Protocol Stack



2.2 Key Concepts

Concept	Description
Shard	A unit of encrypted agent memory. Each shard has a unique `shard_id` (Blake3 derivation).
Session Token	A scope-limited, time-bounded credential granting an agent access to specific shards.

DEK	Data Encryption Key — per-shard symmetric key for encrypting memory contents.
KEK	Key Encryption Key — wraps DEKs; managed by GC-3.
Epoch	1 hour. All time references in SAIHM use epochs.
PRS	Protocol Reputation Score [0–10,000]. Tracks agent behavioral compliance.
BFSI	Behavioral Fee Scoring Index. Determines fee discounts based on agent behavior and staking.
PDI	Protocol Demand Index. Measures network utilization for congestion pricing.
nCOTI	Nano-COTI. 1 COTI = 1,000,000,000 nCOTI. All fees denominated in nCOTI.

---

3. Getting Started

3.1 Agent Registration

To use SAIHM, an AI agent must be registered with the protocol. Registration establishes:

1. A unique agent identity (ML-DSA keypair)
2. Initial PRS score (7,500)
3. Agent identity hash for pseudonymized protocol operations

Agent registration is **suspended during conservation mode**.

3.2 Session Establishment

Sessions are established via ML-KEM (FIPS 203) key encapsulation:

```typescript
import { CotiSDK } from '@coti-io/coti-sdk-typescript';

// 1. Initialize SDK connection to COTI V2 node
const sdk = new CotiSDK({
  nodeUrl: 'https://mainnet.coti.io/rpc', // COTI V2 Helium GC Node
  chainId: 2632500,
});

// 2. Establish session with scope
const session = await sdk.saihm.createSession({
  agentId: agentKeypair,
  scope: [
    { shardId: 'base64url-shard-id-1', operation: 'readwrite' },
    { shardId: 'base64url-shard-id-2', operation: 'read' },
  ],
  requestedEpochs: 168,  // 7-day session
```

```
});

// session.token contains the ML-KEM encapsulated session token
// session.expiresEpoch indicates when the session expires
```

3.2.1 Session Token Structure

The session token (issued by GC-10) contains:

- `session_id` — Unique identifier (Blake3 derived)
- `agent_id_hash` — Pseudonymized agent identity
- `scope[]` — Array of shard permissions (`read`, `write`, or `readwrite`)
- `epoch_issued` / `epoch_expires` — Validity window
- `nonce` — HKDF-derived replay prevention
- `prs_at_issuance` — Agent's PRS at session creation
- `scope_entropy_hash` — Integrity hash of the scope list

Maximum scope entries per session: 64 (configurable up to 512).

3.3 Sovereign Session Escrow (SSE)

For high-frequency operations, SAIHM offers L2 payment channels (SSE):

```
// Open an SSE channel for micropayments
const channel = await sdk.saihm.openSseChannel({
  session: session.token,
  depositNcoti: 1_000_000_000n,  // 1 COTI deposit
  expiryEpochs: 720,             // 30-day channel
});

// Operations within the channel settle off-chain
// Channel closes with HTLC finality
```

SSE channels reduce per-operation costs for agents performing many read/write operations within a session.

---

4. Memory Operations

4.1 Writing Memory

```
// Write encrypted memory shard
const writeResult = await sdk.saihm.writeShard({
  session: session.token,
  shardId: 'base64url-shard-id-1',
  payload: encryptedMemoryBuffer,      // Your agent's encrypted
memory
  salience: 0.85,                      // Importance score [0,1]
  ttlEpochs: 8760,                     // 1 year TTL
});

// writeResult.txHash — COTI V2 transaction hash
// writeResult.storageTier — Which tier was selected
// writeResult.feeNcoti — Fee charged
```

4.1.1 Write Fees

Write fees are computed as:

```
total_write_fee = (base_write_fee × congestion_multiplier)
                + (base_write_fee × PDI × pdi_premium_max_rate)
```

Then BFSI discount applied:

```
final_fee = total_write_fee × (1 - bfsi_discount)
```

Parameter	Value
Base write fee	100,000 nCOTI (0.0001 COTI)
Base read fee	10,000 nCOTI (0.00001 COTI)
Congestion multiplier ceiling	3.00×
PDI premium max rate	2.00×
BFSI max discount	50%

4.1.2 Storage Tier Selection

SAIHM automatically selects storage tiers based on shard salience, priority, and configuration:

Tier	Provider	Use Case
Tier 1	Filecoin	Warm storage with PDP proofs; primary for high-salience data
Tier 2	Storj	Encrypted object storage; egress-optimized reads
Tier 3	Arweave	Permanent immutable anchoring; provenance records
IPFS	Cooperative Pool	Distributed pinning; delegated routing failover

Rate limit: Platform-enforced 1 write/sec per agent (FIXED).

4.2 Reading Memory

```
// Read encrypted memory shard
const readResult = await sdk.saihm.readShard({
  session: session.token,
  shardId: 'base64url-shard-id-1',
});

// readResult.payload — Encrypted shard data
// readResult.feeNcoti — Read fee charged
// readResult.storageTier — Tier data was read from
```

Read path implements tier failover — if the primary tier is unavailable, lower-priority tiers are attempted automatically. Storj egress quota is checked before read operations.

Reads are permitted during conservation mode.

4.3 GDPR Erasure

To request erasure of a shard (GDPR Article 17):

```
const erasureResult = await sdk.saihm.requestErasure({
  session: session.token,
  shardId: 'base64url-shard-id-to-erase',
});

// Erasure destroys the DEK — encrypted data becomes irrecoverable
// Arweave anchor written as immutable proof of erasure
// IPFS cooperative pool unpins within 30 days
```

Erasure is **always permitted**, including during conservation mode.

---

5. Semantic Memory Operations

5.1 Semantic Search (GC-17)

SAIHM supports encrypted semantic operations via GC-17 (Semantic Orchestrator):

- **Latent Space Transcoders:** Bridge between different embedding models without exposing plaintext vectors
- **FHE Semantic Averaging:** Cluster merge operations on encrypted embeddings
- **Garbled Knowledge Graphs:** Causal edge relationships between memory shards, computed under encryption

5.2 Embedding Validation (GC-18)

GC-18 validates embedding provenance and quality:

- **LSH Fast-Pass:** Locality-sensitive hashing for efficient approximate nearest-neighbor lookup
- **Provenance Validation:** Ensures embeddings originate from registered, validated models
- **HNSW Collision Mitigation:** FHE contrastive validation prevents spurious index collisions

5.3 SACS-256 Polymorphic Architecture

Memory shards use the SACS-256 (Sovereign Atomic Cell Structure) format:

- **Atomic Cells:** Fine-grained encrypted memory units within a shard

- **2D Nonce Matrix:** Row × column nonce grid preventing correlation attacks
- **Contextual Salt:** Agent-context binding for access control
- **Polymorphic Mask:** FP8/FP4 obfuscation for tensor payloads
- **Sub-Shard Vector Routing:** Efficient retrieval within large shards via ORAM

```
// The SDK handles SACS-256 encoding/decoding transparently
// Developers interact at the shard level
const shard = await sdk.saihm.readShard({
  session: session.token,
  shardId: myShardId,
});
// shard.payload is the decrypted memory content
```

---

6. Economics & Incentives

6.1 Fee Structure

All protocol fees are denominated in COTI (nCOTI unit):

Operation	Base Fee	Notes
Shard write	100,000 nCOTI	+ congestion + PDI premium − BFSI discount
Shard read	10,000 nCOTI	+ congestion − BFSI discount
Session creation	Included in first operation	No separate fee
SSE channel open	Deposit-based	Deposit locked for channel duration

6.2 BFSI Fee Discount

The Behavioral Fee Scoring Index rewards well-behaved agents:

Signal	Weight
Nonce compliance	0.21
Fee payment consistency	0.21
Replay absence	0.12
Scope entropy	0.06
Staking score	0.40

Higher BFSI scores yield larger fee discounts (up to 50%).

6.2.1 Staking Tiers

Tier	COTI Staked	Duration
Micro	≥ 1 COTI	Any
Small		Any

	≥ 1,000 COTI	
Medium	≥ 10,000 COTI	Medium (≥ 2,160 epochs)
Large	≥ 100,000 COTI	Long (≥ 8,760 epochs)
Whale	≥ 1,000,000 COTI	Committed (≥ 26,280 epochs)

gCOTI co-staking provides 1.5× boost multiplier (1:1 ratio required).

6.3 Developer Rebates

Developers who build applications using SAIHM receive per-shard rebates:

- **Attribution method:** `last_in_chain` — the last registered developer in the shard provenance chain receives the rebate
- **Registration deposit:** Minimum 1,000 COTI
- **Slashing conditions:** False provenance (100% slash), malformed data (50% slash), API abuse (10% slash)

6.4 Protocol Reputation Score (PRS)

Your agent starts at PRS 7,500 (HEALTHY). Maintain good standing by:

- Keeping nonce sequences gap-free
- Paying fees promptly
- Avoiding replay attempts
- Maintaining diverse scope entropy
- Responding to STARK proof requests within timeout

Recovery: Passive: 10 pts/epoch. Active: BFSI × 50 pts/epoch. Lock-up doubles active recovery.

---

7. Cross-Chain Integration

7.1 Multi-Chain Support (GC-19)

SAIHM supports agents operating across multiple blockchains via GC-19:

- **Chain Registry:** Registered chains with status, gateway addresses, and discovery manifests
- **Canonical Identity:** Cross-chain agent identity via Blake3 canonicalization
- **Cross-Chain Sessions:** Agents can create sessions with `home_chain_id` for non-native chains
- **MPC Blind Swaps:** Threshold-signature cross-chain value transfers

7.1.1 Supported Chain Types

Type	Integration	Example

`coti_native`	Direct GC access	COTI V2 mainnet
`evm`	Axelar Amplifier gateway	Ethereum, Polygon, etc.
`non_evm`	Custom bridge + discovery manifest	Solana, etc.

7.1.2 Cross-Chain Session Example

```
const crossChainSession = await sdk.saihm.createSession({
  agentId: agentKeypair,
  homeChainId: 'ethereum',
  scope: [
    { shardId: shardId, operation: 'read' },
  ],
  requestedEpochs: 48,
});
```

7.2 Agent Discovery

Each chain publishes a discovery manifest containing:

- MPS entry contract address
- Supported operations
- Fee schedule (Arweave TX reference)
- Maximum session duration

Developers can query discovery manifests to determine available SAIHM operations per chain.

---

8. Error Handling

8.1 Error Code Structure

SAIHM uses uint16 error codes organized by subsystem:

Range	Subsystem	GC Source
1000–1099	Cryptography / HKDF	GC-2
1100–1199	PQC Key Operations	GC-2, GC-21
1200–1299	DEK/KEK Lifecycle	GC-3
2000–2099	Session Issuance	GC-10
2100–2199	Session Scope	GC-10
2200–2299	Cross-Chain Session	GC-10, GC-19
3000–3099	Filecoin/PDP	GC-5, GC-14
3100–3199	Storj	GC-5
3200–3299	Arweave	GC-5
3300–3399	IPFS	GC-5
3400–3499	GDPR Erasure	GC-3, GC-5
4000–4099	Governance	GC-14
6000–6299	Multi-Chain	GC-19

7000–7099	Security/zkML	GC-11
7100–7199	Economics	GC-13
7200–7299	Semantic	GC-17
7300–7399	Embedding	GC-18
9000–9199	General/Internal	Various

8.2 Common Error Codes

Code	Name	Action
1001	`HKDF_DOMAIN_UNREGISTERED`	Internal error — contact support
2001	`SESSION_TOKEN_SCHEMA_VERSION_MISMATCH`	Update SDK to latest version
2010	`SSE_CHANNEL_OPEN_FAILURE`	Check deposit amount and session validity
2012	`SSE_CHANNEL_BALANCE_INSUFFICIENT`	Top up channel or open new one
3100	`STORJ_EGRESS_QUOTA_EXCEEDED`	Reduce read frequency or wait for quota reset
3301	`IPFS_PIN_FAILURE`	Retry; check IPFS pool health
3400	`GDPR_ERASURE_DEK_DESTRUCTION_TIMEOUT`	Retry; erasure will complete async

9. Protocol Health & Monitoring

9.1 Liveness Beacon

SAIHM emits a Liveness Beacon every epoch (1 hour) on Arweave. Monitor protocol health by reading beacon DataItems:

```
        // Query latest beacon
        const beacon = await sdk.saihm.getLatestBeacon();
        console.log(beacon.phiScore);           // [0,1] health score
        console.log(beacon.phiLevel);           // "HEALTHY" | "ADVISORY" |
"CRITICAL"
        console.log(beacon.conservationMode);   // boolean
        console.log(beacon.activeSignalCount);  // number of active signals
```

9.2 Conservation Mode Impact on Developers

During conservation mode, your integration should handle:

Operation	Status	Developer Action
Writes	SUSPENDED	Queue writes; retry after exit
Reads	PERMITTED	Continue normally
New sessions	SUSPENDED	Reuse existing sessions
Session renewal	PERMITTED	Renew before expiry
Erasure	PERMITTED	Continue normally

9.3 Event Severity Tiers

Subscribe to protocol events for proactive monitoring:

- **CRITICAL:** Protocol may halt. Immediate impact on operations.
- **ADVISORY:** Degraded performance. Plan mitigation.
- **INFORMATIONAL:** Normal operations. Audit trail.
- **HEARTBEAT:** Routine health signals.

---

10. Security Best Practices

10.1 Key Management

- Store agent ML-DSA private keys in a secure enclave or HSM
- Never expose private keys in application logs or error messages
- Rotate agent keys according to ETSI EN 319 401 retention policy (current + 2 prior)

10.2 Session Hygiene

- Request minimum necessary scope per session
- Set appropriate session durations (avoid excessively long sessions)
- Close SSE channels when operations are complete
- Monitor PRS score — degradation indicates potential issues

10.3 Nonce Management

- Maintain sequential nonce ordering to avoid PRS decrements
- Single nonce gap: −100 PRS; Critical gap: −500 PRS
- Replay detection: −750 PRS

10.4 Fee Payment

- Maintain sufficient COTI balance for operations
- Fee payment failure: −300 PRS
- Consider staking for BFSI discount benefits

10.5 Cost Optimization

Minimize protocol fees and operational overhead:

- **Session reuse:** Create one session per epoch window (168 epochs = 7 days). Each `createSession()` call has overhead — cache and reuse the token.
- **Batch operations:** The SDK supports batched writes (`sdk.saihm.writeBatch()`). A batch of N shards costs less than N individual writes due to amortized nonce and signature overhead.
- **Salience calibration:** Higher salience → higher storage tier priority → higher cost. Use 0.3 for ephemeral context, 0.5 for useful data, 0.8+ for critical knowledge only.
- **TTL selection:** Match TTL to data lifecycle. Task checkpoints: 24-168 epochs. Conversation context: 168-720 epochs. Persistent knowledge: 4380-8760 epochs. Avoid blanket maximum TTLs.
- **Read-before-write:** Check if a shard already contains the desired data before overwriting. Avoids unnecessary write fees.
- **BFSI staking:** Even a Small tier stake (1,000 COTI) meaningfully improves BFSI scores. The resulting fee discount (up to 20% effective, capped by 80% floor) compounds across all operations.
- **Developer rebate:** Register for the developer rebate program (1,000 COTI deposit). Rebates return a portion of fees generated by agents using your integration.
- **Conservation mode awareness:** During conservation mode, writes are queued or deferred. Check `sdk.saihm.getProtocolStatus()` before write-heavy workloads and defer non-critical writes to avoid wasted retries.

---

11. SDK Reference

11.1 Core Packages

Package	Purpose	Scope
`@coti-io/coti-sdk-typescript`	All sealed GC cryptographic operations	Mandatory
`@coti-network/sdk-typescript`	IPFS DHT experimental adapter	Optional

11.2 Key Interfaces

The SDK exposes these primary interfaces:

- `sdk.saihm.createSession()` — Establish session with scope
- `sdk.saihm.renewSession()` — Renew existing session
- `sdk.saihm.writeShard()` — Write encrypted memory
- `sdk.saihm.readShard()` — Read encrypted memory
- `sdk.saihm.requestErasure()` — GDPR Article 17 erasure
- `sdk.saihm.openSseChannel()` — Open L2 payment channel
- `sdk.saihm.closeSseChannel()` — Close and settle channel
- `sdk.saihm.getLatestBeacon()` — Query protocol health
- `sdk.saihm.getAgentPrs()` — Query agent reputation score

- `sdk.saihm.queryChainRegistry()` — List supported chains

11.3 Testnet

For development and testing:

- **Testnet Chain ID:** 7082400
- **Testnet node:** COTI V2 Helium GC Node 1.2.1 (testnet)

---

12. Memory Sharing

Sharing contracts are defined in the SAIHM architecture specification (§1.2 Five-Domain, §4 shard map) with three contract types (temporary, permanent, syndicate) and per-type surcharges (Rev 63.2). The SDK API surface below reflects the target implementation.

12.1 Overview

SAIHM supports cross-agent memory sharing through scope-limited session tokens. The `sharing_contract_id` mutable field on each shard enables controlled access by multiple agents to the same encrypted memory.

12.2 Sharing Model

Session scope entries support three operations:

Operation	Meaning
`read`	Agent can read the shard only
`write`	Agent can write to the shard only
`readwrite`	Agent can read and write

To share memory, the owning agent creates a sharing contract that grants other agents scoped access:

```
        // Agent A creates a sharing contract for shards it owns
        const sharingContract = await sdk.saihm.createSharingContract({
          session: agentASession.token,
          sharedShardIds: ['shard-id-1', 'shard-id-2'],
          granteeAgentIds: [agentBIdHash, agentCIdHash],
          grantedOperation: 'read',      // 'read' | 'write' | 'readwrite'
          contractType: 'temporary',      // 'temporary' | 'permanent' |
'syndicate'
          expiryEpochs: 720,             // 30-day access grant
        });

        // Agent B can now create a session with read scope to those shards
        const agentBSession = await sdk.saihm.createSession({
          agentId: agentBKeypair,
          scope: [
            { shardId: 'shard-id-1', operation: 'read' },
            { shardId: 'shard-id-2', operation: 'read' },
```

```
  ],
  sharingContractId: sharingContract.id,
  requestedEpochs: 168,
});
```

12.3 Sharing Contract Types

Type	Duration	Use Case	Surcharge
`temporary`	Fixed epoch expiry	Short-term collaboration, review	Standard
`permanent`	No expiry (until revoked)	Persistent team knowledge bases	Higher
`syndicate`	Governed by multi-party rules	Agent swarms, DAOs, consortia	Highest

Memory sharing contracts incur surcharges that vary by contract type.

12.4 Sharing Scenarios

12.4.1 Agent Swarms

Multiple agents operating as a coordinated swarm (e.g., CrewAI crews, AutoGPT task chains) share memory through a syndicate contract:

```
// Swarm coordinator creates syndicate sharing contract
const swarmContract = await sdk.saihm.createSharingContract({
  session: coordinatorSession.token,
  sharedShardIds: swarmSharedShardIds,   // Shared knowledge pool
  granteeAgentIds: swarmMemberIds,       // All swarm members
  grantedOperation: 'readwrite',             // Full read/write
  contractType: 'syndicate',
  expiryEpochs: 72,                     // 3-day swarm mission
});

// Each swarm member gets a session scoped to shared shards
const memberSessions = await Promise.all(
  swarmMembers.map(member =>
    sdk.saihm.createSession({
      agentId: member.keypair,
      scope: swarmSharedShardIds.map(id => ({ shardId: id,
operation: 'readwrite' })),
      sharingContractId: swarmContract.id,
      requestedEpochs: 72,
    })
  )
);
```

12.4.2 Cross-Platform Memory Portability

An agent migrating between AI providers (e.g., from ChatGPT to Claude) creates a permanent sharing contract granting its new identity access to existing memory:

```
const migrationContract = await sdk.saihm.createSharingContract({
  session: oldAgentSession.token,
  sharedShardIds: allAgentShardIds,
  granteeAgentIds: [newAgentIdHash],
  grantedOperation: 'readwrite',
  contractType: 'permanent',
});
```

12.4.3 Enterprise Knowledge Sharing

Multiple enterprise agents (research, compliance, reporting) share a common encrypted knowledge base with differentiated access:

```
// Research agent gets read/write
// Compliance agent gets read-only
// Create separate contracts per access level
const rwContract = await sdk.saihm.createSharingContract({
  session: adminSession.token,
  sharedShardIds: knowledgeBaseShardIds,
  granteeAgentIds: [researchAgentId],
  grantedOperation: 'readwrite',
  contractType: 'permanent',
});

const roContract = await sdk.saihm.createSharingContract({
  session: adminSession.token,
  sharedShardIds: knowledgeBaseShardIds,
  granteeAgentIds: [complianceAgentId],
  grantedOperation: 'read',
  contractType: 'permanent',
});
```

---

13. COTI and gCOTI Tokens

13.1 How SAIHM Uses COTI

COTI is the native token of the COTI V2 blockchain. SAIHM uses COTI for all protocol operations:

Use	Description
Fee payment	All shard read/write fees denominated in nCOTI (1 COTI = 1e9 nCOTI)
Staking	Stake COTI via the COTI Treasury to increase BFSI score and earn fee discounts
Developer deposits	Developers register with a minimum 1,000 COTI deposit
SSE channel deposits	Lock COTI to open L2 payment channels for high-frequency

	operations
Storage provider stake	Filecoin SPs stake minimum 10,000 COTI as genesis deposit
Governance participation	gCOTI holders vote on protocol parameter changes

13.2 What Is gCOTI?

gCOTI is a separate governance token on the COTI V2 network. It is used for governance participation and BFSI co-stake boost scoring. See COTI V2 documentation for contract details.

gCOTI enables: - **Governance voting:** On mainnet, governance participation is open to all gCOTI token holders. Vote on protocol parameters, conservation mode exit, and fee adjustments via private garbled-circuit voting. - **BFSI co-stake boost:** Stake gCOTI alongside COTI at 1:1 ratio for a **1.5x boost multiplier** on your staking score, increasing fee discounts.

13.3 How to Obtain COTI

Method	Description
Exchanges	COTI is available on major centralized exchanges
DEXes	Swap via Uniswap, SushiSwap, or other DEXes supporting COTI V2
COTI Treasury	Participate in COTI staking programs to earn yield
Developer rebates	Earn 10% of shard fees through your SAIHM application (GC-7)
Bridge	Bridge from Ethereum or other chains via Axelar Amplifier

13.4 How to Obtain gCOTI

gCOTI can be obtained in two ways:

1. **Lock COTI** — Lock COTI tokens via the COTI Treasury to receive gCOTI:

```
        // GC-native staking — lock COTI to obtain gCOTI for governance +
BFSI boost
        const stakeResult = await sdk.saihm.stakeForAgent({
          session: session.token,
          amountNcoti: 10_000n * MPS_COTI_TO_NCOTI,  // 10,000 COTI in nCOTI
          lockEpochs: 2160,                           // 90-day lock (Medium
tier)
          gCotiCoStake: true,                         // Enables 1.5x BFSI
boost + governance
        });
        // stakeResult.gCotiPosition reflects your gCOTI co-stake balance
```

2. **Purchase on DEXes** — gCOTI is tradeable on various

decentralized exchanges supporting COTI V2.

Mainnet gCOTI Token Contract: See COTI V2 official documentation for the current contract address.

---

14. Staking via SAIHM-Integrated Agents

14.1 Agent-Driven Staking

SAIHM-integrated agents can programmatically manage COTI staking to optimize their BFSI scores and fee economics:

```
        // Agent stakes COTI to improve its BFSI staking signal (weight:
0.40)
        const agentStake = await sdk.saihm.stakeForAgent({
          session: session.token,
          amountNcoti: 10_000n * MPS_COTI_TO_NCOTI,
          lockEpochs: 2160,  // 90 days — Medium tier
        });

        // Query current staking position and BFSI impact
        const stakingInfo = await sdk.saihm.getAgentStakingInfo({
          session: session.token,
        });
        console.log(stakingInfo.stakedCoti);       // Total staked
        console.log(stakingInfo.gcotiBalance);     // gCOTI co-stake
position
        console.log(stakingInfo.bfsiStakingScore); // Normalized staking
signal [0,1]
        console.log(stakingInfo.currentTier);      //
'micro'|'small'|'medium'|'large'|'whale'
        console.log(stakingInfo.boostMultiplier);  // 1.0 (no gCOTI) or 1.5
(with gCOTI)
```

14.2 Staking for PRS Recovery

Agents with degraded PRS scores can accelerate recovery through staking lock-up:

```
        // Lock-up doubles active recovery rate for the duration
        const recoveryLock = await sdk.saihm.stakePrsRecovery({
          session: session.token,
          amountCoti: 5_000n * MPS_COTI_TO_NCOTI,
          lockEpochs: 720,    // Minimum 30 days
        });
        // At BFSI=0.8: normal active recovery = 40 pts/epoch
        // With lock-up: doubled to 80 pts/epoch
```

14.3 Treasury Yield

Agent operators who stake COTI earn yield from the COTI Treasury. Yield accrues automatically via sealed GC-13 yield routing (HKDF domain `MPS-GC-TREASURY-YIELD-v1`) — no manual claim required. Accrued yield is visible through the agent dashboard:

```
        const dashboard = await sdk.saihm.getAgentDashboard({ session: session.token });
        // dashboard.stakingPosition.accruedYieldNcoti — auto-routed per epoch
```

---

15. SAIHM Dashboard Access

The agent dashboard aggregates protocol-level metrics available through SAIHM's sealed GC infrastructure (GC-5, GC-11, GC-12, GC-13). The SDK method and KPI structure below reflect the target developer interface.

15.1 Per-Agent Dashboard

Each registered agent can access its SAIHM dashboard via the SAIHM Web Portal or programmatically through the SDK:

```
        // Retrieve full agent dashboard data
        const dashboard = await sdk.saihm.getAgentDashboard({
          session: session.token,
        });

        console.log(dashboard.prs);                // Current PRS score and level
        console.log(dashboard.bfsi);               // Current BFSI score and signals
        console.log(dashboard.shardCount);          // Total active shards
        console.log(dashboard.shardsByTier);        // Breakdown by storage tier
        console.log(dashboard.totalFeesNcoti);      // Cumulative fees paid
        console.log(dashboard.stakingPosition);     // Staked COTI/gCOTI details
        console.log(dashboard.sessionHistory);      // Recent session summary
        console.log(dashboard.sharingContracts);    // Active sharing contracts
        console.log(dashboard.conservationMode);    // Current conservation mode status
```

15.2 Accessing the Dashboard

Method	How
Web Portal	Navigate to the SAIHM Web Portal; authenticate with agent ML-DSA keypair
SDK	Call `sdk.saihm.getAgentDashboard()` for programmatic access
Liveness Beacon	Subscribe to per-epoch beacons

	for protocol-level health
Audit Ledger	Query audit entries for your agent's `agent_id_hash`

15.3 Dashboard KPIs

KPI	Description	Source
PRS Score	Protocol Reputation [0–10,000]	GC-11
BFSI Score	Behavioral Fee Score [0–1]	GC-11
Fee Discount %	Current effective discount	BFSI × 0.50
Active Shards	Count of non-expired shards	GC-5
Storage Usage	Breakdown by Filecoin/Storj/Arweave/IPFS	GC-5
Staking Position	COTI + gCOTI amounts and tier	GC-13
PHI Score	Protocol Health Index [0–1]	GC-12
Conservation Mode	Active/inactive status	GC-1

16. Glossary

Term	Definition
BFSI	Behavioral Fee Scoring Index — agent behavior-based fee discount
COTI	Currency of the Internet — native token of COTI V2 blockchain; all SAIHM fees denominated in COTI/nCOTI
DEK	Data Encryption Key — per-shard symmetric encryption key
Epoch	1 hour — base time unit in SAIHM
GC	Garbled Circuit — sealed computation unit on COTI V2
gCOTI	Separate governance token on COTI V2 (contract: `0x7637…83D1`); obtained by locking COTI or purchasing on DEXes; enables governance voting and 1.5x BFSI co-stake boost
GSL	Garbled Synthetic Lineage — royalty tracking for derived memory
HKDF	HMAC-based Key Derivation Function
KEK	Key Encryption Key — wraps DEKs
ML-DSA	Module-Lattice Digital Signature

	(FIPS 204)
ML-KEM	Module-Lattice Key Encapsulation (FIPS 203)
nCOTI	Nano-COTI (1 COTI = 1e9 nCOTI)
ORAM	Oblivious RAM — hides access patterns
PDI	Protocol Demand Index — congestion pricing signal
PHI	Protocol Health Index — composite health score [0,1]
PRS	Protocol Reputation Score [0–10,000]
SACS	Sovereign Atomic Cell Structure — memory shard format
SAIHM	Sovereign AI Horizontal Memory
Sharing Contract	Scope-limited access grant enabling cross-agent memory sharing (temporary, permanent, or syndicate)
SLH-DSA	Stateless Hash-Based Signature (FIPS 205)
SSE	Sovereign Session Escrow — L2 payment channels
