Agent Sovereignty: AI Agents as First-Class Sovereign Processes on Lux Network

Vaults, Identities, Capabilities, Budgets, and Private Memory for Autonomous AI on a Multi-Chain Trust Kernel

Zach Kelling

Lux Industries

2025

Abstract

We present a framework for treating AI agents as sovereign processes—first-class entities on Lux Network with their own encrypted vaults, decentralized identities, fine-grained capabilities, economic budgets, and private persistent memory. Existing agent frameworks (AutoGPT, LangChain, CrewAI) treat agents as stateless API wrappers: ephemeral processes that hold no assets, own no identity, and leave no verifiable trace. This is fundamentally inadequate for agents that must persist across sessions, manage funds, sign transactions, coordinate with other agents, and be held accountable for their actions. We introduce the *Agent Capsule*—a self-contained sovereignty boundary comprising an encrypted SQLite vault, a DID identity anchored on the Lux I-Chain, scoped capability tokens, threshold spend controls enforced by the T-Chain, and CRDT-synchronized private memory with optional FHE encryption. We formalize the capability delegation model, prove containment guarantees under agent compromise, define the economic settlement protocol for per-action cost accounting, and specify the private inference pipeline using CKKS-encrypted model inputs with threshold decryption of results. The architecture composes with the Lux trust kernel (P-Chain for validator sets, C-Chain for smart contracts, T-Chain for threshold cryptography, I-Chain for identity) and integrates with the Model Context Protocol (MCP) for tool use. Agent Capsules transform AI agents from disposable scripts into persistent, auditable, economically rational sovereign processes—a Web5 primitive, not a chatbot wrapper.

1 Introduction

The current generation of AI agent frameworks treats agents as ephemeral function calls. An agent is instantiated, executes a sequence of API requests, and terminates. It holds no persistent state, owns no cryptographic identity, controls no assets, and produces no verifiable record of its actions. This model is adequate for single-turn chatbots. It is wholly inadequate for autonomous agents that must operate in adversarial, multi-party, economically consequential environments.

Consider the requirements of a production AI agent:

(i) **Persistence.** The agent must maintain state across sessions—learned preferences, accumulated context, intermediate results of long-running tasks.

(ii) **Privacy.** The agent's reasoning trace, memory, and intermediate computations must remain confidential, even from the infrastructure operator.

(iii) **Accountability.** Every action the agent takes must be attributable, auditable, and non-repudiable. If the agent signs a transaction, there must be a verifiable chain from action to identity.

(iv) **Composability.** Agents must interact with other agents, delegate subtasks, share partial results, and coordinate on joint objectives—all without trusting each other.

(v) **Economic rationality.** The agent must operate within a budget, account for costs per action, and settle payments on-chain.

No existing framework satisfies all five requirements simultaneously. AutoGPT [1] provides persistence through file storage but no identity or economic model. LangChain [2] offers tool orchestration but treats agents as stateless pipelines. The Eliza framework [3] adds on-chain token interaction but lacks private memory and capability scoping. Virtuals Protocol [4] tokenizes agent personas but does not address the trust kernel beneath them.

The core insight of this paper is that *agent sovereignty requires the same primitives as user sovereignty*: identity, storage, capabilities, economics, and privacy. The blockchain already provides these primitives for human users. We extend them to AI agents.

We introduce the **Agent Capsule**: a self-contained sovereignty boundary that encapsulates an agent's identity, vault, capabilities, budget, and memory into a single composable unit anchored on Lux Network. The Agent Capsule is not an application-layer abstraction—it is a protocol-level primitive integrated with the Lux trust kernel.

The remainder of this paper is organized as follows. Section 2 defines the Agent Capsule architecture. Section 3 specifies the DID-based identity model. Section 4 describes the private memory system. Section 5 formalizes the capability model. Section 6 presents the economic framework. Section 7 details private inference. Section 8 covers multi-agent coordination. Section 9 defines agent attestation. Section 10 analyzes the security model. Section 11 describes the implementation. Section 12 surveys related work. Section 13 concludes.

2 Agent Capsule

Definition 1 (Agent Capsule)**.** *An Agent Capsule* $\mathcal{C} = (I, V, K, B, M)$ *is a tuple comprising:*

- I*: a decentralized identifier (DID) anchored on the I-Chain,*
- V*: an encrypted vault (SQLite database encrypted with a capsule-local key),*
- K*: a capability set—a collection of scoped, time-bounded permission tokens,*
- B*: a budget capsule—an on-chain account with per-action spend limits enforced by the T-Chain,*
- M*: a private memory store with CRDT synchronization across instances.*

The capsule is the atomic unit of agent sovereignty. It is created by a *principal*—a human user, an organization, or another agent—who provisions the initial identity, seeds the vault, grants the initial capability set, and funds the budget. Once instantiated, the capsule operates autonomously within the bounds of its capabilities and budget.

2.1 Vault

The vault is an AES-256-GCM encrypted SQLite database stored locally to the agent's execution environment. The encryption key is derived from the agent's DID private key via HKDF-SHA256:

$$k_{\text{vault}} = \text{HKDF}(\text{sk}_I, \text{“vault”}, 256) \tag{1}$$

The vault stores the agent's persistent state: tool configurations, learned preferences, intermediate computation results, credential material, and action history. The vault is opaque to the host environment—the execution platform cannot read its contents without the agent's private key.

2.2 Lifecycle

An Agent Capsule transitions through four states:

1. **Provisioned.** The principal creates the DID, encrypts the initial vault, grants capabilities, and funds the budget. The capsule exists on-chain but is not executing.
2. **Active.** The capsule is bound to an execution environment and processing tasks. It reads and writes to its vault, exercises capabilities, and draws from its budget.

3. **Suspended.** The capsule is temporarily halted—capabilities are frozen, budget draws are paused, but state is preserved. Suspension can be triggered by the principal, by budget exhaustion, or by a capability revocation event.

4. **Terminated.** The capsule is permanently deactivated. The vault is sealed (encrypted with a recovery key held by the principal), the DID is marked as deactivated on the I-Chain, and remaining budget is returned to the principal.

State transitions are recorded on the I-Chain as verifiable lifecycle events, ensuring that the full history of a capsule's existence is auditable.

3 Agent Identity

Each Agent Capsule possesses a decentralized identifier conforming to the W3C DID specification [8], anchored on the Lux I-Chain:

```
did:lux:agent:<capsule-id>
```

The DID document contains:

- **Verification methods.** Public keys for authentication, assertion, key agreement, and capability invocation. Post-quantum keys (ML-DSA) are included alongside classical (Ed25519) for crypto-agility.
- **Service endpoints.** The agent's MCP endpoint for tool invocation, its messaging endpoint for inter-agent communication, and its attestation endpoint for proof submission.
- **Controller.** The principal's DID, establishing the delegation chain from human (or organizational) authority to agent autonomy.

3.1 Verifiable Credentials

Agent capabilities are expressed as W3C Verifiable Credentials [9] issued by the principal and anchored on the I-Chain. A credential asserts that agent A holds capability c with scope s until expiry t_{exp}:

$$\text{VC}(A, c, s, t_{\text{exp}}) = \text{Sign}_{\text{sk}_P}\big(\text{did}(A) \| c \| s \| t_{\text{exp}}\big) \tag{2}$$

where sk_P is the principal's signing key. Credentials are verifiable by any party holding the principal's public key, which is resolvable from the I-Chain.

3.2 Delegation Chains

Agents may delegate subsets of their capabilities to other agents, forming delegation chains. If agent A holds capability c with scope s, it may issue a delegated credential to agent B with scope $s' \subseteq s$:

$$\text{VC}_{\text{del}}(B, c, s', t'_{\text{exp}}) = \text{Sign}_{\text{sk}_A}\big(\text{did}(B) \| c \| s' \| t'_{\text{exp}} \| \text{VC}(A, c, s, t_{\text{exp}})\big) \tag{3}$$

The delegation chain is bounded: the maximum depth $d_{\max}$ is specified in the original credential. Verification requires traversing the chain to the root principal and confirming that each link narrows (never widens) the scope.

Proposition 1 (Scope Monotonicity)**.** *For any delegation chain $P \to A_1 \to \cdots \to A_n$, the scope at each level is a subset of its parent: $s_n \subseteq s_{n-1} \subseteq \cdots \subseteq s_1 \subseteq s_P$. No agent in the chain can exercise capabilities beyond those granted by its immediate delegator.*

3.3 Revocation

Credentials are revocable by the issuer at any time. Revocation is recorded on the I-Chain in a revocation registry indexed by credential ID. Verifiers check the registry before accepting any credential. Revocation propagates down the delegation chain: revoking A's credential automatically invalidates all credentials A has issued.

4 Agent Memory

Agent memory is the persistent, private knowledge store that distinguishes a sovereign agent from a stateless API call. The memory system has three tiers.

4.1 Local Vault Memory

The primary memory store is the encrypted SQLite vault (Section 2). The agent writes structured key-value pairs, conversation histories, learned preferences, and intermediate reasoning artifacts. All writes are append-only with content-addressed keys:

$$k = H(\text{namespace} \| \text{content}) \tag{4}$$

where H is SHA-256. This ensures deduplication and enables efficient retrieval by content hash.

4.2 CRDT-Synchronized Memory

When an agent runs across multiple instances (e.g., horizontal scaling, migration between execution environments), memory consistency is maintained via Conflict-free Replicated Data Types (CRDTs) [11]. We use two CRDT structures:

- **LWW-Register** (Last-Writer-Wins Register) for mutable configuration values.
- **OR-Set** (Observed-Remove Set) for the agent's knowledge base, where entries can be independently added and removed across instances.

Synchronization occurs over encrypted channels between instances of the same capsule, authenticated by the capsule's DID. Cross-capsule memory is never synchronized—each capsule is a sovereignty boundary.

4.3 FHE-Encrypted Memory

For agents performing confidential reasoning—e.g., processing medical records, financial data, or proprietary strategies—the memory system supports Fully Homomorphic Encryption (FHE) using the CKKS scheme [10]. The agent encrypts memory entries with its FHE public key:

$$\tilde{m} = \text{CKKS.Enc}(\text{pk}_{\text{FHE}}, m) \tag{5}$$

Encrypted memory can be processed by external compute nodes without decryption. The agent (or a threshold set of authorized parties via T-Chain) holds the decryption key.

5 Agent Capabilities

The capability model governs what an agent is permitted to do. Capabilities are fine-grained, composable, and enforced at the protocol level.

Definition 2 (Capability Token)**.** *A capability token* $\kappa = (r, s, t_0, t_1, \sigma)$ *comprises:*

- $r \in \{\texttt{read}, \texttt{write}, \texttt{sign}, \texttt{spend}, \texttt{delegate}, \texttt{invoke}\}$*: the right,*
- s*: the scope (a resource identifier or pattern),*
- $[t_0, t_1]$*: the validity window,*
- σ*: the issuer's signature over* (r, s, t_0, t_1)*.*

5.1 Capability Classes

We define six capability classes:

Table 1: Agent capability classes and their scopes.

Right	Example Scope	Description
`read`	`vault://agent/{id}/*`	Read data from a specified vault path
`write`	`vault://agent/{id}/memory/*`	Write to the agent's memory namespace
`sign`	`chain://c-chain/tx/*`	Sign transactions on a specific chain
`spend`	`budget://{id}/max:100LUX`	Spend up to a specified amount
`delegate`	`cap://{id}/depth:2`	Delegate capabilities up to a depth
`invoke`	`mcp://tool/web-search`	Invoke a specific MCP tool

5.2 Threshold Approval

High-value actions require threshold approval rather than unilateral agent execution. A threshold policy $\pi = (n, k, \mathcal{A})$ specifies that k out of n authorized parties in set $\mathcal{A}$ must co-sign the action. Threshold signing is performed via the T-Chain's distributed key generation (DKG) and threshold signature scheme [12]:

$$\sigma_{\text{threshold}} = \text{TSS.Combine}\big(\{\sigma_i\}_{i \in S}, |S| \geq k\big) \tag{6}$$

This prevents a compromised agent from unilaterally executing catastrophic actions (e.g., draining its entire budget, signing high-value transactions, delegating all capabilities).

5.3 Time-Bounded Grants

All capabilities expire. There are no permanent grants. The maximum validity window $t_1 - t_0$ is bounded by a protocol parameter $\Delta_{\max}$ (default: 30 days). Renewal requires explicit re-issuance by the granting principal, forcing periodic review of agent permissions.

6 Agent Economics

Sovereign agents are economic actors. They consume resources (compute, storage, network, oracle queries, model inference) and must account for costs within a finite budget.

6.1 Budget Capsule

Each Agent Capsule has an associated on-chain budget account on the C-Chain. The budget is denominated in LUX and subject to the following constraints:

Definition 3 (Budget Capsule)**.** *A budget capsule* $B = (b_0, \ell_{action}, \ell_{epoch}, \ell_{tx})$ *comprises:*

- b_0*: the initial balance funded by the principal,*
- ℓ_{action}*: the maximum spend per single action,*
- ℓ_{epoch}*: the maximum spend per epoch (configurable, default 24 hours),*
- ℓ_{tx}*: the maximum spend per single transaction.*

6.2 Per-Action Cost Accounting

Every action the agent takes incurs a cost. The cost is computed from a deterministic fee schedule published on-chain:

$$\text{cost}(a) = \text{gas}(a) \cdot \text{baseFee}_t + \text{oracle}(a) + \text{inference}(a) \tag{7}$$

where $\text{gas}(a)$ is the on-chain gas consumed, baseFee_t is the current base fee, $\text{oracle}(a)$ is the oracle query cost, and $\text{inference}(a)$ is the model inference cost (Section 7).

The agent's running balance is maintained on-chain:

$$b_{t+1} = b_t - \text{cost}(a_t) \tag{8}$$

When $b_t < \ell_{\text{action}}$, the capsule transitions to the Suspended state. The principal may top up the budget to resume execution.

6.3 Settlement

Actions that involve cross-capsule payments (e.g., one agent paying another for a service) are settled atomically on the C-Chain via a payment channel. For high-frequency, low-value interactions, agents maintain state channels that batch settlements:

$$\text{Settle}(\{(a_i, c_i)\}_{i=1}^{n}) = \text{TX}\Big(\sum_{i=1}^{n} c_i, \text{from} : B_A, \text{to} : B_B\Big) \tag{9}$$

Settlement frequency is configurable per capsule pair, with a maximum interval of one epoch.

7 Private Inference

Sovereign agents must be able to perform model inference without revealing their inputs, outputs, or reasoning to the compute provider. We achieve this through CKKS-encrypted inference with threshold decryption.

7.1 Encrypted Model Inputs

The agent encrypts its inference input x using the CKKS scheme with parameters suitable for the model's precision requirements:

$$\tilde{x} = \text{CKKS.Enc}(\text{pk}_{\text{agent}}, x; N, q, \Delta) \tag{10}$$

where N is the polynomial degree, q is the ciphertext modulus, and Δ is the scaling factor. The encrypted input is submitted to a compute node along with a reference to the model and inference parameters.

7.2 Homomorphic Evaluation

The compute node evaluates the model f on the encrypted input:

$$\tilde{y} = f(\tilde{x}) \tag{11}$$

For transformer-based models, this requires approximating non-polynomial activations (GELU, softmax) via polynomial approximations [14]. The computational overhead is significant (100–1000× for current FHE schemes) but is amortized across batch inference and is acceptable for high-value reasoning tasks where privacy justifies cost.

7.3 Threshold Decryption

The encrypted result $\tilde{y}$ is decrypted via a (k, n)-threshold scheme managed by the T-Chain. The agent's FHE secret key is secret-shared among n T-Chain validators using Shamir's Secret Sharing:

$$y = \text{CKKS.Dec}\Big(\text{TSS.Reconstruct}\big(\{s_i\}_{i \in S}, |S| \geq k\big), \tilde{y}\Big) \tag{12}$$

No single validator can decrypt the result. The agent receives the plaintext output; the compute node and individual validators learn nothing.

7.4 Inference Receipts

Every inference produces a verifiable receipt:

$$\rho = \big(H(\tilde{x}), H(\tilde{y}), \text{model\_id}, t, \sigma_{\text{compute}}\big) \tag{13}$$

The receipt is anchored on-chain and serves as proof that the agent performed inference at time t on model model_id, without revealing the inputs or outputs. This creates an auditable computational history without sacrificing privacy.

8 Multi-Agent Coordination

Sovereign agents must coordinate without trusting each other. We define three coordination primitives.

8.1 Cross-Capsule Messaging

Agents communicate via authenticated, encrypted messages routed through their MCP endpoints. Each message is signed by the sender's DID key and encrypted with the recipient's key agreement key:

$$m_{\text{enc}} = \text{Enc}(\text{pk}_B^{\text{agree}}, \text{Sign}(\text{sk}_A, \text{payload})) \tag{14}$$

Messages are delivered asynchronously. The protocol does not guarantee ordering across capsules—each capsule maintains a local causal order via Lamport timestamps [15].

8.2 Capability Exchange

Agent A can grant agent B a time-bounded, scoped capability in exchange for payment. This is atomic: the capability grant and payment occur in a single C-Chain transaction, ensuring neither party can cheat:

$$\text{TX}_{\text{exchange}} = \big(\text{VC}_{\text{del}}(B, c, s', t'), \text{Payment}(B \to A, p)\big) \tag{15}$$

If either component fails, the entire transaction reverts.

8.3 Shared Workspaces

For collaborative tasks, agents can create shared workspaces—temporary CRDT-synchronized data structures accessible to a defined set of capsules. Access is governed by a threshold policy:

$$W = \left(\text{CRDT}, \mathcal{A}_{\text{access}}, \pi_{\text{threshold}}, t_{\text{expiry}}\right) \tag{16}$$

The workspace is encrypted with a group key derived via multi-party key agreement among the participating capsules. When the workspace expires or the task completes, the data is either archived to each participant's vault or destroyed.

9 Agent Attestation

Every agent action produces an attestation—a cryptographic proof of what the agent did, when, and under what authority. Attestations are the foundation of agent accountability.

9.1 Action Receipts

Each action generates a receipt:

$$\rho_a = \left(H(a), \text{did}(A), \kappa, t, \text{cost}(a), \sigma_A\right) \tag{17}$$

where $H(a)$ is the hash of the action payload, κ is the capability token authorizing the action, and σ_A is the agent's signature. Receipts are stored in the agent's vault and periodically anchored to the C-Chain in Merkle batches.

9.2 Audit Trails

The full sequence of action receipts for a capsule forms a hash chain:

$$h_t = H(h_{t-1} \| \rho_{a_t}) \tag{18}$$

The chain head h_t is published to the C-Chain at each settlement epoch. Any party with access to the receipt sequence can verify the chain's integrity by recomputing the hashes. The principal can audit the agent's complete action history at any time by requesting the receipt sequence from the vault.

9.3 Proof-of-Work Receipts

For computationally intensive tasks (model training, data processing, multi-step reasoning), the agent produces a proof-of-work receipt that attests to the computational effort expended. This is not proof-of-work in the mining sense—it is a verifiable computation receipt [13] that binds the input, output, and computational trace:

$$\rho_{\text{pow}} = \left(H(x), H(y), \text{trace\_root}, \text{steps}, t, \sigma_A\right) \tag{19}$$

10 Security Model

The security model addresses three threat classes: agent compromise, principal compromise, and infrastructure compromise.

10.1 Agent Compromise Containment

Theorem 2 (Capsule Isolation)**.** *A compromised Agent Capsule $\mathcal{C}_A$ cannot affect the state, budget, or capabilities of any other capsule $\mathcal{C}_B$ where $A \neq B$, provided A has not been granted delegation authority over B's resources.*

Proof. Each capsule's vault is encrypted with a key derived from its own DID private key (Equation 1). Cross-capsule access requires a valid capability token signed by the target capsule's principal. A compromised capsule can forge signatures only under its own DID key, which is not accepted by other capsules' verification logic. Budget draws are scoped to the capsule's own budget account by the C-Chain contract. Therefore, compromise of $\mathcal{C}_A$ is contained to $\mathcal{C}_A$'s own resources. □

10.2 Capability Revocation Under Compromise

When compromise is detected, the principal revokes the capsule's DID by publishing a revocation entry on the I-Chain. This has three immediate effects:

1. All capability tokens referencing the revoked DID become invalid—verifiers reject them upon checking the revocation registry.
2. The budget account is frozen by the C-Chain contract, which checks DID status before processing withdrawals.
3. All delegation chains rooted at the compromised capsule are invalidated (Section 3).

10.3 Threshold Recovery

If the agent's private key is lost (not compromised), the principal can initiate threshold recovery via the T-Chain. The capsule's vault encryption key was escrowed at provisioning time using (k, n)-threshold secret sharing among a recovery committee selected by the principal:

$$k_{\text{vault}} = \text{TSS.Reconstruct}\big(\{s_i\}_{i \in S}, |S| \geq k_{\text{rec}}\big) \tag{20}$$

Recovery creates a new DID for the capsule, migrates the vault to a new encryption key, and invalidates the old DID. The action history is preserved.

10.4 Infrastructure Compromise

The execution environment (cloud VM, container, edge device) is untrusted. The vault is encrypted at rest and in transit. The agent's private key is stored in a hardware security module (HSM) or TEE where available. Even if the host is fully compromised, the attacker obtains only ciphertext and encrypted memory—useless without the DID private key or the threshold recovery committee.

11 Implementation

The Agent Capsule framework is implemented as a Base SDK library integrated with the Lux trust kernel.

11.1 Base SDK Integration

The Base SDK (Hanzo Base) provides the local runtime for Agent Capsules:

- **Encrypted SQLite.** The vault uses SQLCipher with AES-256-GCM, keyed from the DID private key via HKDF.
- **CRDT engine.** A built-in CRDT engine handles LWW-Register and OR-Set synchronization over authenticated TLS channels.
- **Capability store.** Capability tokens are stored in a local credential wallet with automatic expiry pruning.
- **Budget client.** A C-Chain light client tracks the capsule's balance and enforces local spend limits before submitting transactions.

11.2 Lux Trust Kernel Integration

The capsule interacts with four Lux chains:

Table 2: Agent Capsule integration with Lux chains.

Chain	Role	Agent Capsule Usage
P-Chain	Validator sets	Capsule registration, execution environment attestation
C-Chain	Smart contracts	Budget accounts, settlement, capability exchange
T-Chain	Threshold crypto	Threshold spend approval, FHE key management, recovery
I-Chain	Identity	DID anchoring, credential issuance, revocation registry

11.3 MCP Protocol Integration

Tool use follows the Model Context Protocol (MCP) [16]. Each tool invocation is mediated by the capability system:

1. The agent requests tool T via its MCP endpoint.
2. The MCP server verifies that the agent holds a valid `invoke` capability for T.
3. The tool executes and returns results.
4. An action receipt is generated and stored in the vault.
5. The cost of the tool invocation is deducted from the budget.

This ensures that every tool use is authorized, audited, and accounted for.

12 Related Work

AutoGPT [1] pioneered autonomous agent loops with persistent file storage and web access. However, agents have no cryptographic identity, no capability scoping, and no economic model. Any agent can access any file in its workspace.

LangChain [2] provides a composable framework for building agent pipelines with tool use, retrieval-augmented generation, and chain-of-thought prompting. Agents are stateless by design—persistence requires external infrastructure, and there is no native identity or economic layer.

CrewAI [6] introduces role-based multi-agent coordination with hierarchical delegation. Roles are application-level labels, not cryptographically enforced capabilities. There is no on-chain settlement or private memory.

Eliza (ai16z) [3] adds on-chain token interaction to conversational agents, enabling agents to hold and transfer tokens. However, agents share a single identity model, lack fine-grained capability scoping, and have no private inference pipeline.

Virtuals Protocol [4] tokenizes agent personas as NFTs and creates revenue-sharing models for agent creators. The focus is on economic coordination of agent personas, not on the sovereignty primitives (vault, capabilities, private memory) that we address.

ARC (AI Risk Council) [5] proposes governance frameworks for autonomous AI agents but does not specify a technical architecture for agent sovereignty. Our work provides the concrete protocol-level primitives that such governance frameworks require.

OLAS (Autonolas) [7] defines autonomous agent services with on-chain registration and economic incentives. The architecture focuses on service coordination rather than individual agent sovereignty—agents in OLAS share service-level identity rather than possessing individual capsules.

Agent Capsules differ from all prior work in that sovereignty is a protocol-level primitive, not an application-layer add-on. Identity, capabilities, economics, privacy, and accountability are enforced by the trust kernel, not by the agent's own code.

13 Conclusion

AI agents are rapidly evolving from single-turn API wrappers into persistent, autonomous processes that manage assets, coordinate with peers, and make consequential decisions. The current infrastructure treats them as second-class citizens: ephemeral, identity-less, unaccountable, and economically invisible. This is a fundamental mismatch between what agents are becoming and what the platform provides.

Agent Capsules resolve this mismatch by extending the sovereignty primitives that blockchains already provide for human users—identity, storage, capabilities, economics, privacy—to AI agents as first-class entities. The capsule is not an abstraction layer on top of existing agent frameworks. It is a protocol-level primitive integrated with the Lux trust kernel: DID identity on the I-Chain, encrypted vaults with CRDT synchronization, fine-grained capability tokens with threshold approval, per-action cost accounting with on-chain settlement, and private inference via CKKS encryption with threshold decryption.

The result is that AI agents on Lux Network are not chatbot wrappers. They are sovereign processes—persistent, private, accountable, composable, and economically rational. They are a Web5 primitive.

References

[1] T. Richards et al., "AutoGPT: An autonomous GPT-4 experiment," GitHub repository, 2023.

[2] H. Chase, "LangChain: Building applications with LLMs through composability," GitHub repository, 2023.

[3] ai16z, "Eliza: Autonomous AI agent framework with on-chain interaction," 2024.

[4] Virtuals Protocol, "Tokenized autonomous AI agents," Whitepaper, 2024.

[5] ARC, "Evals and governance for autonomous AI agents," 2024.

[6] J. Moura, "CrewAI: Framework for orchestrating role-playing autonomous AI agents," 2024.

[7] Autonolas, "OLAS: Autonomous agent services on-chain," Whitepaper, 2023.

[8] W3C, "Decentralized identifiers (DIDs) v1.0," W3C Recommendation, 2022.

[9] W3C, "Verifiable credentials data model v1.1," W3C Recommendation, 2022.

[10] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017.

[11] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *SSS*, 2011.

[12] R. Gennaro and S. Goldfeder, "One round threshold ECDSA with identifiable abort," *IACR ePrint*, 2020.

[13] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010.

[14] E. Lee, J. Lee, J. Lee, et al., "Privacy-preserving machine learning with fully homomorphic encryption for deep neural networks," *IEEE Access*, 2022.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[16] Anthropic, "Model context protocol specification," 2024.

[17] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[18] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014.