Proof-of-Descendancy: Identity for Self-Replicating LLM Agents

A Blockchain-Based Framework for Verifiable Agent Lineage in OpenClaw

Mohab Ahmed m.ahmed-22@student.tudelft.nl

Johan Pouwelse J.A.Pouwelse@tudelft.nl

Bulat Nasrulin B.Nasrulin@tudelft.nl

Andy Zaidman A.E.Zaidman@tudelft.nl

TU Delft Delft University of Technology

1 Motivation & Problem

- **Self-replicating OpenClaw agents** need more than ordinary public-key identity.
- **Autonomous agents** can **create child agents** that run independently, use their **own keys,** and participate as **separate peers**. In this setting, proving control over a key is not enough: a verifier also needs to know whether the agent was created by an **authorized parent**, belongs to a **trusted lineage**, is not revoked, and has the requested capability.
- Existing approaches such as centralized identity providers, classical PKI, SSI, HD wallets, and blockchain commitments **each solve part of the problem,** but none directly prove that a self-replicating agent is an **authorized descendant of a trusted origin.**
- **Core problem:**
  OpenClaw-like networks **cannot reliably distinguish legitimate descendants** from spoofed, cloned, replayed, or unauthorized agents using key possession alone.

2 Research Questions

How can **proof-of-descendancy** be used to construct a secure, unified cryptographic **identity framework** for **self-replicating Open-Claw agents?**

Sub questions

1) How can hierarchical deterministic key architectures support **verifiable multi-generational agent lineage?**

2) To what extent do parent-signed certificates and Bitcoin-anchored commitments **mitigate spoofing and unauthorised replication?**

3) How can agents maintain a **unified identity** across networking, financial, and authorisation domains while **preserving parent key secrecy?**

3 System Design

Proof-of-descendancy verifies agent legitimacy through lineage, not key ownership alone.

The framework combines five components:

1. HD key separation

Each agent family **starts from root material** and derives **separate hardened branches** for identity authority, IPv8 networking, Bitcoin anchoring, encryption, and operational signing. HD derivation supports **compartmentalization**, but does not itself prove lineage.

2. Parent-signed child certificates

Each **parent signs a certificate** for the child it creates. This certificate binds the child's identity, public keys, capabilities, validity period, revocation reference, and parent-child relationship. This is the **primary authorization evidence**.

3. Merkle-batched Bitcoin anchoring

Certificate hashes are batched **into a Merkle tree** and the **Merkle root is anchored** as a public commitment. Bitcoin anchoring provides **tamper-evident evidence,** but does not authorize agents by itself.

4. Lineage-chain verification

A **verifier checks** the chain of parent-signed certificates back to a **trusted Genesis agent**. Each step must be **signed by the correct parent** and linked to the next generation.

5. IPv8/OpenClaw admission

During admission, the child **proves live key possession** and presents its **proof bundle**. OpenClaw can then accept or reject the agent based on lineage, anchoring, revocation, expiry, capabilities, and freshness.

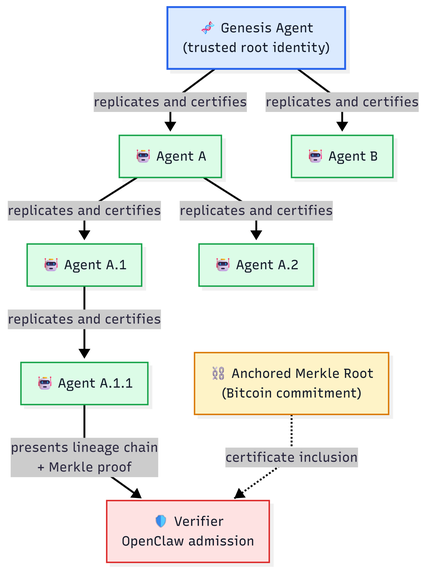

Figure 1: Self-replication family tree

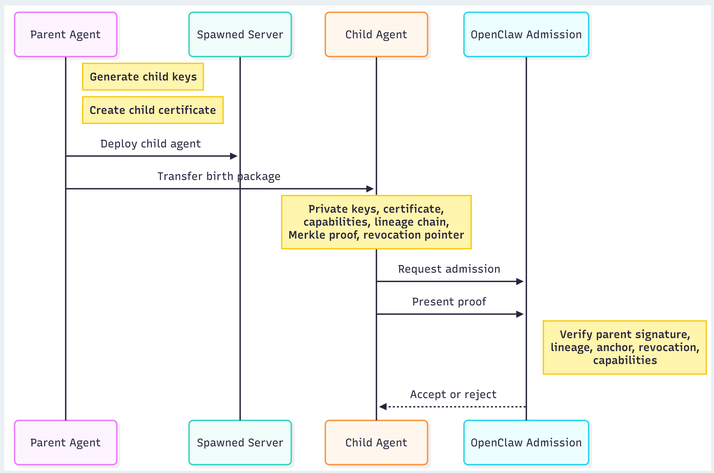

Figure 2: Self-replication flow

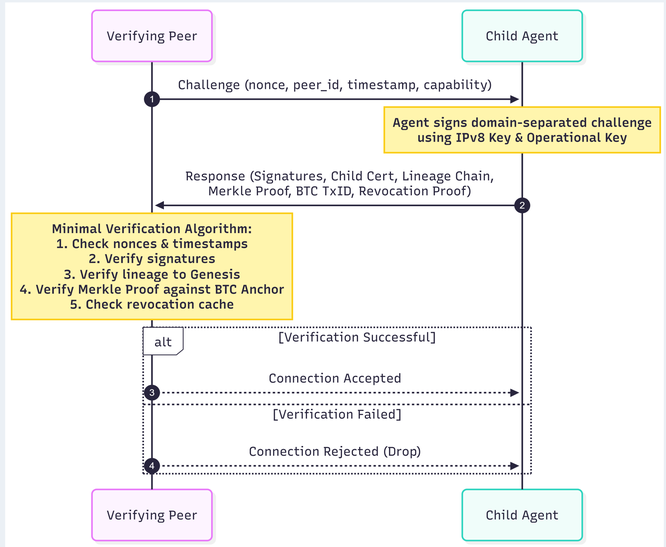

Figure 3: IPv8 Handshake flow

4 Evaluation & Results

Evaluation questions

- **EQ1:** Does the verifier accept valid multi-generational lineage proofs?
- **EQ2:** Does it reject malformed, tampered, expired, revoked, unanchored, or unauthorized proofs?
- **EQ3:** How do proof size and verification latency scale with lineage depth?
- **EQ4:** Can lineage verification affect OpenClaw admission decisions?

Standalone verifier results

- Tested lineage depths: **1, 2, 4, 8, 16, 32**
- Valid-proof trials: **180**
- Adversarial mutation trials: **450**
- Result: **180/180 valid proofs accepted**
- Result: **450/450 adversarial proofs rejected**
- Proof size and verification latency increased predictably with lineage depth.

OpenClaw admission results

Lineage mode	Trials	Accepted	Flagged/rejected
Disabled	42	42	0
Optional	42	42	39
Required	42	6	39

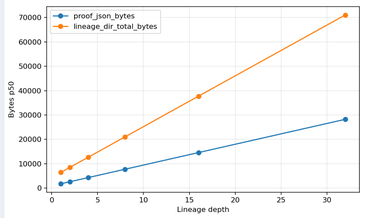

Figure 4: Storage usage by lineage depth

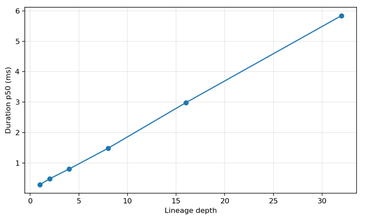

Figure 5: Verification latency by lineage depth

Required mode enforced lineage-aware admission: valid baseline trials were admitted, while adversarial cases were rejected. This shows that proof-of-descendancy **can be connected to the runtime** path where OpenClaw decides whether an agent may join.

5 Limitation & Future Work

Limitations

- The **Bitcoin anchoring backend was mocked**; no real transaction publication, fee dynamics, mempool behavior, block inclusion, or chain reorganization effects were evaluated.
- The adversarial suite covered selected malformed, revoked, expired, unanchored, missing-capability, and replay-style cases, but **not every possible protocol or encoding attack.**
- The system was **not evaluated under production-scale** IPv8/OpenClaw deployment, high concurrency, large batches, or denial-of-service load.
- A valid lineage proof does **not guarantee benign agent behavior** or safe tool use.

Future work

- Replace mock anchoring with **real Bitcoin anchoring.**
- Add **privacy-preserving lineage proofs** and selective disclosure.
- Evaluate the system in a **production IPv8/OpenClaw deployment**.
- **Test adversarial load** and denial-of-service pressure.

DELFTCLAW **DelftClaw repository**