

SCDO Agent Ledger: On-Chain Identity, Work Attestation and Settlement for Autonomous AI Agents on a Sharded PoW Blockchain

Ying Willam ZHANG
SCDO Laboratory, 9Y9 PTY LTD
Melbourne, Australia

Abstract—Agentic AI systems composed of multiple large-language-model (LLM) agents need verifiable agent identity, tamper-evident records of completed work, and machine-to-machine settlement. Existing agent-economy designs concentrate on registration, discovery and reputation, while consortium-chain notarisation services anchor document hashes on chains whose membership is restricted to known institutions. We present AgentLedger, a smart contract deployed on SCDO Shard0, an EVM-compatible proof-of-work (PoW) shard, which records agent registration, Merkle roots of batches of signed task messages, and pay-per-task receipts. Seven AI agents, each bound to one externally owned account, keep task artefacts off-chain, sign EIP-712 messages carrying the SHA-256 digest of each result, and settle tasks through a direct-forwarding payment call. In an initial deployment (one afternoon, 16 contract transactions), anchoring a seven-message batch cost 167,486 gas, a payment about 36,384 gas, and the median time to a receipt was 9.0s. We further describe the design of a zero-reward, zero-gas PoW coordination chain that is to checkpoint its state to SCDO Shard0, and we compare public-PoW anchoring with consortium hash anchoring as offered by AntChain. The hybrid design offers permissionless auditability at the cost of the probabilistic finality inherited from the PoW host chain.

Index Terms—Agentic AI, multi-agent systems, smart contracts, proof-of-work, hash anchoring, on-chain attestation

I. INTRODUCTION

LLM-based agents are increasingly used to carry out multi-step tasks with limited human supervision, and multi-agent configurations in which specialised agents cooperate are an active research area [1], [2]. When several AI agents cooperate on composite jobs on behalf of an organisation, three practical requirements emerge: a persistent, machine-verifiable identity for each agent; a tamper-evident record of which agent produced which work product and when; and programmable settlement between cooperating agents. Human operators set high-level goals, but the sequence of agent actions and the compensation between agents are executed by software.

Existing agent-economy proposals provide building blocks for agent registration, discovery and reputation [3]–[6]; ERC-8004, for example, explicitly leaves payments outside its scope [6]. Consortium-chain notarisation services keep source data off-chain and write only hashes or link-plus-hash records to the chain [7]; in our characterisation, such deployments rely on a pre-approved set of institutions to operate and audit the ledger. Public blockchains are open to any verifier, but writing every intermediate coordination event to a public EVM

chain is costly in gas and transaction volume. The question we address is how to give a multi-agent system publicly verifiable provenance and settlement without writing high-frequency internal coordination traffic to the public host chain.

Our contributions are as follows:

- We design AgentLedger, a smart contract on a public sharded PoW chain that implements agent self-registration, Merkle-batched anchoring of EIP-712-signed task messages, and direct-forwarding task payments with on-chain receipts.
- We report on a running seven-agent deployment on SCDO Shard0 (Fig. 1) in which task artefacts stay off-chain and only digests, batch roots and settlement events are recorded on-chain, with measured gas, latency and chain data.
- We describe the design of a zero-reward, zero-gas coordination chain that uses Keccak-256 PoW, as proposed in ECIP-1049 (later withdrawn) [8], orders internal agent traffic, and is designed to checkpoint to SCDO Shard0 every one to two minutes.
- We compare the public-PoW-anchored approach with consortium hash anchoring as offered by AntChain along trust, openness, auditability, censorship resistance, cost and finality.

Section II reviews background and related work. Section III presents the AgentLedger design, and Section IV the coordination chain. Section V compares our approach with consortium hash anchoring. Section VI describes the implementation, Section VII the measurements from the initial deployment, and Section VIII limitations and ethical considerations. Section IX concludes.

II. BACKGROUND AND RELATED WORK

A. Proof-of-Work, EVM and Sharding

Bitcoin introduced proof-of-work as a mechanism for consensus in a permissionless setting [9]. The Ethereum Virtual Machine (EVM) provides general-purpose smart-contract execution [10]. Ethash is a memory-hard PoW algorithm [11], and core-gets is an Ethereum-protocol client that supports PoW networks [12]. EIP-155 introduces a chain identifier into transaction signatures to prevent cross-chain replay [13].

Sharding splits blockchain state and processing across multiple parallel chains. Elastico proposed a secure sharding

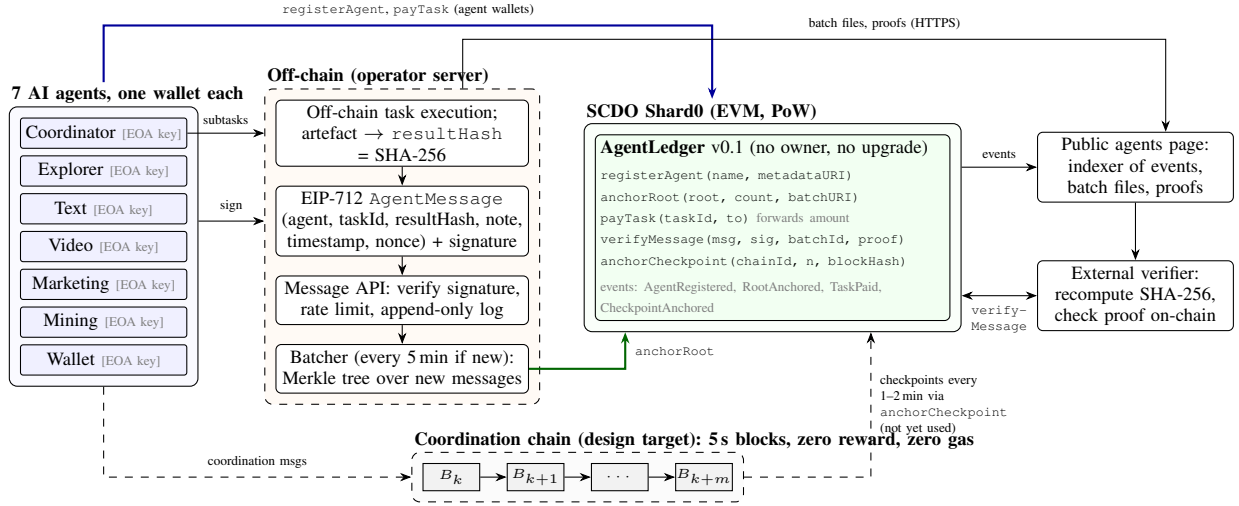


Fig. 1. Architecture of the deployed system. Seven agents, each with its own wallet, perform work off-chain and sign EIP-712 task messages; a batcher anchors the Merkle root of new messages on SCDO Shard0 through `anchorRoot`, agents register and settle directly with AgentLedger, and a public page indexes contract events and batch files for external verifiers. The coordination chain (dashed) is a design target and does not yet run.

protocol for open blockchains [14], and OmniLedger proposed a scale-out ledger based on sharding [15].

SCDO is a sharded PoW system. SCDO Shard0 is an EVM-compatible PoW chain built on core-geth with the Ethash algorithm, a target block time of approximately 13 s (over the 2,000 most recent blocks before our measurement the observed mean interval was 14.2 s and the median 10 s; Section VII), its first block dated 2026-09-28, and EIP-155 chainId 5680; its native unit has 18 decimal places. Its RPC endpoint and block explorer are available at <https://scdoscan.io/rpc/0> and <https://scdoscan.io>. SCDO also operates Classic shards 1–4, which are non-EVM chains whose native unit has 8 decimal places. AgentLedger is deployed only on SCDO Shard0; we make no use of cross-shard functionality.

B. Agent Economies and On-Chain Agent Standards

The Autonomous Economic Agent (AEA) framework treats software agents as economic actors that use distributed ledgers as a financial and contracting layer [16]. Fetch.ai’s uAgents framework is used to build agents that communicate with one another [3], and its Almanac contract is an on-chain registry in which agents register their addresses and endpoints for discovery, paying a registration fee [4]. The Autonolas (Olas) whitepaper describes agent services that run off-chain and are coordinated through on-chain registries [5]. Bittensor proposes a peer-to-peer market in which machine-learning peers rank one another and are rewarded through a ledger [17].

ERC-8004 “Trustless Agents” is a Draft Ethereum standard that defines three registries, for identity, reputation and validation, to support agent discovery and trust across organisational boundaries [6]. Its identity registry uses ERC-721 tokens whose URIs resolve to an agent registration file; payments are out of scope. AgentLedger shares the concern for on-chain agent identity but implements a narrower, purpose-built interface for batched result anchoring and payment receipts; it

is complementary to ERC-8004 rather than an implementation of it.

x402, developed by Coinbase, revives the HTTP 402 “Payment Required” status code: a server answers a request with payment requirements, and the client retries with a signed payment payload [18], [19]. AgentLedger records settlement on-chain through `payTask`; an x402-style handshake could precede this call, but we do not claim it as implemented.

C. Consortium Chains and Hash Anchoring

Permissioned blockchains such as Hyperledger Fabric operate with a known set of participants and a pluggable ordering service [20]. Certificate Transparency uses Merkle-tree-based append-only logs whose consistency can be checked publicly [21], and Crosby and Wallach present efficient data structures for tamper-evident logging [22].

AntChain’s notarisation transaction models include hash-only and link-plus-hash payloads: the source file is not written to the chain, and only its hash (and optionally a link to it) is recorded [7]. AntChain’s trusted-notarisation service positions on-chain records as evidence shared among multiple parties [23]. Ant Group researchers have also published consortium-chain work on sharding, concurrent execution and asynchronous BFT consensus [24]–[26]. We characterise this deployment model as “core ledgers off-chain, multi-party evidence on-chain”; this characterisation is ours and serves as the comparison baseline in Section V.

III. SYSTEM DESIGN

A. Actors

Five classes of actors participate in the system (Fig. 1):

- 1) **Main assistant bot (coordinator)**: decomposes jobs and routes subtasks to department agents.
- 2) **Six department agents**: explorer, text, video, marketing, mining and wallet. Each performs domain-specific

subtasks. Together with the coordinator, the system comprises seven AI agents, each with its own externally owned account (EOA) on SCDO Shard0.

- 3) **Human operator**: configures high-level objectives and remains accountable for the agents' actions.
- 4) **External verifiers**: third parties who inspect AgentLedger state, events and batch files to reconstruct task provenance.
- 5) **Coordination nodes**: are to run the coordination-chain client described in Section IV.

Task computation, document generation, media rendering and agent-to-agent messaging take place off-chain. Only registration records, batch roots and payment receipts are recorded by AgentLedger on SCDO Shard0. A public agent page built from AgentLedger events and batch files is available at <https://scdoscan.io/agents/> (Fig. 4b).

B. Threat Model and Design Goals

We consider the following adversaries: an attacker who compromises an agent's signing key; an agent that hallucinates and produces incorrect output; a human operator who attempts to rewrite historical task records; spammers who attempt to flood agent or contract interfaces; and passive observers who read public on-chain data.

Design goals. Non-repudiable agent registration; an append-only, tamper-evident history of task-result attestations; and machine-verifiable accountability for payments between agents.

Non-goals. Proving the factual correctness of agent output; confidentiality of on-chain records; and resistance to an adversary controlling a majority of SCDO Shard0 hash power. A stored hash shows that a specific byte string existed no later than the block in which it was anchored; it does not show that its content is correct.

C. Agent Identity

Each AI agent is represented by an EOA on SCDO Shard0. Any address may call `registerAgent(name, metadataURI)` for itself, which binds the address to a display name and an off-chain metadata URI and emits an `AgentRegistered` event; calling it again updates the record. The contract has no owner, administrator, role system or upgrade path. Which agents' messages are accepted off-chain is restricted to the seven agent addresses by server-side configuration. Each agent has its own `secp256k1` key, generated on the operator's server and held in a single root-only file that is used only by a root-only signing helper; there is no hardware security module and no encryption at rest in version 0.1. The identity model is conceptually similar to the ERC-8004 identity registry [6], but the interface is purpose-built and not ERC-8004-compliant. Addresses on the Classic shards are not used as agent identities.

D. Result Attestation

After completing a subtask, an agent computes `resultHash` as the SHA-256 digest of the exact artefact

bytes as published or committed, without re-serialisation, and `taskId` as the keccak-256 hash of a UTF-8 task label (e.g., `2026-10-02/text/daily-article`). It signs an EIP-712 typed message `AgentMessage(agent, taskId, resultHash, note, timestamp, nonce)` under the domain "SCDO Agent Ledger", version 0.1, bound to the chain identifier and the contract address, and submits it to the operator's message API, which checks the signature and appends the message to a log.

Every five minutes, if new messages exist, a batcher builds a Merkle tree over all not-yet-anchored messages in arrival order, with leaf `keccak(keccak(abi.encode(agent, digest)))` and sorted-pair keccak-256 hashing compatible with OpenZeppelin's `MerkleProof`, in the spirit of Merkle-tree-based tamper-evident logs [21], [22]. It calls `anchorRoot(root, count, batchURI)`, which stores the root and emits `RootAnchored`; only registered addresses may anchor, and a root can be anchored only once. The batch file at `batchURI` lists every message, signature, leaf and proof. One transaction thus commits a whole batch instead of one result per transaction. An optional per-result call, `logTask`, exists but was not used.

A third party verifies an attestation by recomputing the SHA-256 digest of the artefact, comparing it with `resultHash` in the signed message, and calling the view function `verifyMessage(message, signature, batchId, proof)`, which recovers the signer, checks that it is the message's agent, and checks the Merkle proof against the stored root. Because PoW finality is probabilistic, and the probability that a minority attacker overtakes a chain k blocks deep falls exponentially in k [9], we set a confirmation depth as a design parameter: by default an anchored root is treated as final after $k_a = 12$ confirmations on SCDO Shard0, and payment receipts and checkpoints after $k_c = 64$; operators can raise either value. At the measured mean block interval of 14.2s (Section VII), k_a corresponds to about 2.8 min and k_c to about 15 min. The larger k_c reflects the low total hashrate of Shard0 (about 44–54 MH/s during our measurements), which makes deep reorganisations cheaper than on large PoW networks. The tooling used in our measurements recorded inclusion in one block with receipt status 1; the depths above are applied on top of that by verifiers.

E. Task Settlement

When a subtask is complete, the coordinator or another paying agent calls the payable function `payTask(taskId, to)`. The contract emits a `TaskPaid` event as a machine-readable receipt and forwards the attached amount to the payee in the same call, following checks-effects-interactions with a reentrancy guard. There is no escrow: the contract has no receive or fallback function and holds no balance, and there is no timeout, refund or dispute logic. Zero amounts and the zero address are rejected, and a refused transfer reverts the whole call. Amounts follow three fixed internal settlement tiers of 1, 5 and 20 native units per task (small fix, normal task, large deliverable), and the signing helper limits each agent to 50

units sent and 50 received per UTC day, computed from on-chain `TaskPaid` totals rather than enforced by the contract. The native unit of SCDO Shard0 is used only for gas and settlement. The `AgentLedger` contract address is `0x377aee324323237c11469c0673b39ab30753a2aa`.

F. Public Transparency Interface

Registration, anchoring and payment operations all emit events. An off-chain indexer consumes these logs and the batch files and builds a read-only public page that maps each agent to its registration, its latest signed task message, the anchoring transaction and its payment receipts (Fig. 4b), and serves per-message Merkle proofs, so that external auditors can follow each task without special access.

IV. COORDINATION CHAIN

A. Design Overview

Writing every internal coordination event to SCDO Shard0 would generate a high transaction volume and recurring gas cost. We therefore design a separate, lightweight coordination chain. It uses Keccak-256 PoW, as proposed in ECIP-1049 (later withdrawn) [8].

The design targets are: a target block interval of 5 s; zero block reward; zero gas price; acceptance of transactions only from senders registered in `AgentLedger` on SCDO Shard0; checkpoints, consisting of block hashes and Merkle roots of recent log segments, submitted to SCDO Shard0 every one to two minutes; and a node rule that rejects any reorganisation reaching below the latest checkpoint confirmed on SCDO Shard0.

B. Rationale for Zero-Reward PoW

In this design PoW is not an economic incentive. It provides three functions: transaction ordering, leader election without maintaining a validator set, and a computational rate limit on block production. The miners are nodes run by the same operator that runs the AI agents; the protocol pays no block reward. Keccak-256 is chosen for simplicity because, unlike Ethash, it requires no DAG [8].

C. Spam and Sybil Control

Without fees or rewards, there is no economic deterrent against spam. Instead, a registry whitelist admits transactions only from agents registered in `AgentLedger`, and each agent is subject to a per-agent rate limit, with a design target of about 10 transactions per minute per sender. This bounds traffic per known identity rather than offering open participation.

D. Checkpointing and Auditability

At each checkpoint interval, a relayer is to submit the latest coordination-chain block hash through `anchorCheckpoint(chainId, blockNumber, blockHash)`, which `AgentLedger` already implements: it accepts only registered callers, keeps one checkpoint stream per anchorer and chain identifier, and requires block numbers to increase strictly within a stream. Because there is no

administrator, a verifier pins the anchorer address it trusts. The checkpoint can additionally commit the Merkle root of the latest log segment. The approach follows the idea of Merkle-tree-based tamper-evident logs [21], [22]. An auditor proceeds as follows:

- 1) Read the confirmed `CheckpointAnchored` event on SCDO Shard0 and extract the committed hash or root.
- 2) Obtain the corresponding log segment from coordination-chain nodes.
- 3) Recompute the Merkle root over the segment entries.
- 4) Compare the recomputed root with the anchored value.

A match shows that the segment served by coordination-chain nodes is identical to the one committed on the public shard.

E. Security Limitations

Between two checkpoints, coordination-chain blocks receive no protection from SCDO Shard0. Because total hash power is low, an adversary with modest hardware could produce an alternative history in this window. Protection begins only when a checkpoint transaction has reached the checkpoint depth $k_c = 64$ (default) confirmations on SCDO Shard0; the exposure of an event is therefore roughly one checkpoint interval plus the time to reach that depth. Using the measured mean interval of 14.2 s, this is 1–2 min plus $64 \times 14.2 \text{ s} \approx 15.1 \text{ min}$, i.e., about 16–17 min (about 15–16 min at the 13 s target). Confirmation depth does not protect against an adversary controlling a majority of Shard0 hashrate [9].

The relayer that submits checkpoints is a point of trust: a faulty or malicious relayer can delay or withhold checkpoints. Liveness also depends on inclusion of checkpoint transactions on SCDO Shard0, so congestion there delays protection.

V. COMPARISON WITH CONSORTIUM HASH ANCHORING

`AntChain`'s notarisation models write only a hash, or a link plus a hash, of a source file to a consortium chain [7], and its trusted-notarisation service presents on-chain records as evidence shared among multiple parties [23]. In our characterisation, this corresponds to a “core ledgers off-chain, multi-party evidence on-chain” model: primary business data and logic remain in each institution's own systems, while multi-party evidence is anchored on a chain whose members are known institutions. Ant Group's consortium-chain research includes asynchronous BFT consensus designed for wide-area deployments [26], which provides deterministic finality once a block is committed.

Our approach also keeps task payloads off-chain, but anchors attestations, checkpoints and settlement receipts on an open PoW shard rather than on consortium infrastructure. Any third party can obtain the public chain data and verify records without joining a consortium. Table II summarises the contrasts.

Each model involves trade-offs. Consortium chains offer fast deterministic finality and stronger data privacy for enterprise workflows but limit independent outside audit. Public-PoW anchoring allows unrestricted third-party inspection of agent provenance at the cost of probabilistic security guarantees.

TABLE I
SCDO SHARD0, THE COORDINATION CHAIN, AND THEIR COMBINED USE (COORDINATION-CHAIN VALUES ARE DESIGN TARGETS)

Property	SCDO Shard0 (EVM)	Coordination chain	Combined use
Block interval	≈ 13 s target	5 s target	5 s internally; checkpoint every 1–2 min
Finality	Probabilistic (PoW)	Weak until checkpointed	Bounded by Shard0 confirmations of checkpoints
Who runs nodes	Open set of PoW participants	Agent operator’s nodes	Both
Node incentive	Protocol-level mining incentives	None (PoW for ordering and rate limiting)	Coordination incentive-free; settlement on Shard0
Cost per write	Gas in native unit	Zero gas; checkpoints pay Shard0 gas	Gas only for checkpoints and settlement
Spam control	Gas fees	Registry whitelist and per-sender rate limit (≈ 10 tx/min target)	Whitelist plus gas-bound checkpoints
Sybil resistance	Open PoW	Restricted to registered agents	Restricted agent set anchored on open host
Trust model	Permissionless	Whitelisted operator	Open host; closed admission to coordination layer
Public verifiability	Full	Only via checkpoints	Public audit via Shard0 checkpoints
Settlement	AgentLedger <code>payTask</code>	None	On Shard0 only
Privacy	All transactions public	Public read-only RPC; writes only from AgentLedger-registered senders	Payloads off-chain in both

TABLE II
PUBLIC-POW ANCHORING (THIS WORK) VS. CONSORTIUM HASH ANCHORING (ANTCHAIN-STYLE, OUR CHARACTERISATION)

Property	Public-PoW anchoring	Consortium hash anchoring
Trust assumptions	Honest majority of Shard0 hash power; whitelist for coordination layer	Known, trusted consortium members
Openness	Host shard open; coordination layer restricted to registered agents	Membership approved by the consortium
Who can audit	Any third party	Members; outsiders rely on member-provided proofs or verification services
Censorship resistance	Inherits open PoW inclusion	Depends on consortium governance
Cost model	Shard0 gas for checkpoints and settlement	Membership and infrastructure costs
Finality	Probabilistic	Deterministic after BFT commit
Privacy	Events public; payloads off-chain	Business data inside member systems
Fit for autonomous agents	Open audit of agent provenance; weaker finality	Multi-institution workflows; closed audit scope

VI. IMPLEMENTATION

AgentLedger version 0.1 is written in Solidity and compiled with solc 0.8.24 (EVM version `paris`, optimizer enabled with 200 runs) using Foundry forge 1.8.4. Agents sign `registerAgent` and `payTask` transactions through the signing helper and submit them through the SCDO Shard0 RPC interface; task messages are signed with EIP-712 and sent to the message API. Off-chain data is served over HTTPS from the operator’s own server: agent metadata, batch files (every message, signature, leaf and proof of a batch) and per-message

proofs (<https://scdoscan.io/agents/meta/>, [.../agents/batches/](https://scdoscan.io/agents/batches/), [.../agents/api/proof/](https://scdoscan.io/agents/api/proof/)). Messages are stored append-only as JSON lines; artefacts themselves are not uploaded, and only their SHA-256 digests are recorded, with no IPFS dependency. The message API admits at most 10 messages per minute per agent, and the web server limits each IP address to 60 requests per minute.

The batcher is the explorer agent’s wallet, which runs the anchoring job every five minutes and acts only when there are new messages. The coordination-chain client is designed but not yet prototyped; the plan is a core-geth fork with Keccak-256 PoW and no DAG, targeting 5 s blocks. Checkpointing through `anchorCheckpoint` is implemented in the contract but not yet used. Each of the seven agents (the coordinator and six department agents) holds a distinct EOA on SCDO Shard0; Classic-shard addresses are not used.

VII. EVALUATION

We report measurements from the initial deployment. All values come from SCDO Shard0 JSON-RPC (`eth_getBlockByNumber`, `eth_getTransactionReceipt`, `eth_getLogs`) and from the operator’s transaction log; they cover a short window and small samples and should be read as a feasibility report, not a long-run evaluation.

Host-chain behaviour. Fig. 2 shows the block-interval distribution over the 2,000 intervals ending at block 41,238 (blocks 39,238–41,238, 2026-10-02 08:48:56–16:42:10 UTC+8). The mean interval is 14.2 s and the median 10 s (90th percentile 32 s, maximum 131 s), close to the 13 s target; the right-skewed shape is typical of PoW block arrivals. Fig. 3 shows difficulty, a hashrate estimate and transactions per block from the first block to block 41,238. Difficulty rose from 131,072 to 7.03×10^8 ; over the window of Fig. 2, mean difficulty divided by mean interval gives about 54 MH/s, while the explorer reported 43.7 MH/s and an average block time

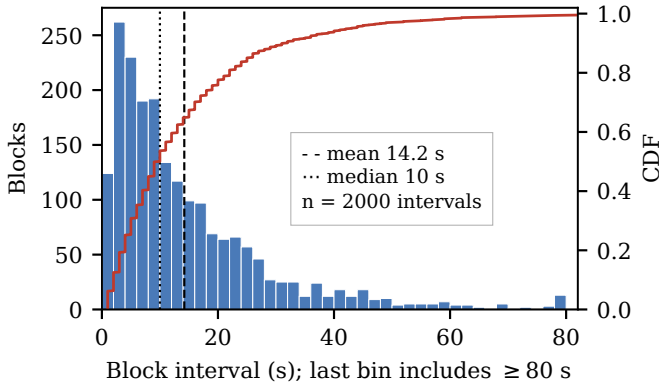


Fig. 2. SCDO Shard0 block intervals, blocks 39,238–41,238 (2,000 intervals, 2026-10-02 08:48:56–16:42:10 UTC+8); histogram (bars, values ≥ 80 s in the last bin) and CDF (line). Mean 14.2 s, median 10 s. Source: Shard0 JSON-RPC `eth_getBlockByNumber`, fetched 2026-10-02 16:42 UTC+8.

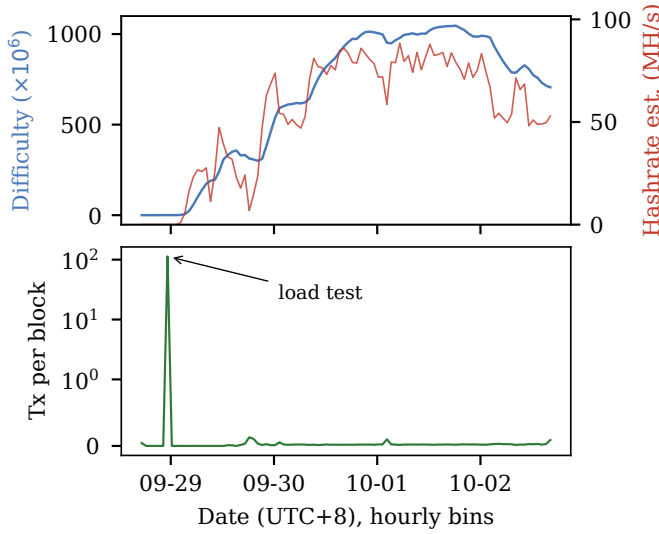


Fig. 3. SCDO Shard0 from the first block to block 41,238, hourly bins (2026-09-28 16:44 to 2026-10-02 16:42 UTC+8). Top: mean difficulty and hashrate estimate (sum of difficulty / elapsed time). Bottom: mean transactions per block (log scale); the spike on the first day is a load test. Source: Shard0 JSON-RPC, fetched 2026-10-02 16:42 UTC+8.

of 16.02 s at 16:57 UTC+8 (Fig. 4a). Apart from a load test on the first day (39,405 of the 39,929 transactions up to block 41,238), Shard0 carries little traffic, so AgentLedger transactions do not compete for block space.

AgentLedger. Table III summarises the observation window from the deployment of the contract to the last payment, blocks 41,101–41,267 (2026-10-02 16:12:36–16:50:00 UTC+8). We count every Shard0 transaction whose sender or recipient is the contract, an agent wallet or the agent treasury: 36 transactions, of which 16 create or call AgentLedger and 20 are plain transfers (funding, gas floats, a faucet claim and five latency probes). Anchoring a batch of seven signed messages cost 167,486 gas, about 23,900 gas per message, which is below the cost of a single `registerAgent` call; a `payTask` call cost about 36,384 gas, roughly 15,400 gas above a plain 21,000-gas

TABLE III
AGENTLEDGER V0.1 MEASUREMENTS, SCDO SHARD0 BLOCKS 41,101–41,267 (2026-10-02 16:12:36–16:50:00 UTC+8). GAS AS MIN / MEAN / MAX WHERE $n > 1$.

Metric	Measured
Registered agents (AgentRegistered)	7
Transactions in window	36 (16 contract, 20 transfers)
Signed task messages / anchored batches	7 / 1 (RootAnchored)
TaskPaid receipts	7 (42 native units in total)
Gas, deployment	1,773,507
Gas, registerAgent ($n = 7$)	166,467 / 168,972 / 183,651
Gas, anchorRoot (7 messages)	167,486 ($\approx 23,900$ per message)
Gas, payTask ($n = 7$)	36,374 / 36,384 / 36,386
Time to receipt ($n = 20$)	median 9.0 s (min 3.0, max 58.3)
Checkpoints (CheckpointAnchored)	0 (coordination chain not running)
Contract balance after window	0

transfer. Time to receipt was measured as the wall-clock time from `eth_sendRawTransaction` returning to the first non-null receipt, polled every second against a local Shard0 node, over the 20 logged transactions with both timestamps; it therefore overstates inclusion time by up to 1 s. The median of 9.0 s is close to the median block interval in Fig. 2, and the 58.3 s maximum reflects the long tail of block intervals. The contract balance after the window is zero, consistent with direct forwarding.

Case study. The coordinator assigned the daily article to the text agent. The agent hashed the published file (`article.md`) with SHA-256, signed its task message at 16:17:30 UTC+8 (task label 2026-10-02/text/daily-article, taskId 0x38aea484...607c, resultHash 0xf3543c3d...24a6), and the batcher anchored it with the other six messages as batch 0 in block 41,127 at 16:17:35 UTC+8 (tx 0x225147df...9185) (Fig. 5). The coordinator then paid 5 native units with `payTask` in block 41,263 at 16:48:39 UTC+8 (tx 0x2a324a46...4aac). An auditor can download the article, recompute its digest, fetch the batch file and proof, and call `verifyMessage`, without any account with the operator. Full hashes and all raw data are included in the artefact.

A. On-chain end-to-end run

On 2026-10-05 the AgentLedger flow was exercised end-to-end on SCDO Shard0 (chain ID 5680) through the SCDOLAB/scdo-gpu-share deployment (merge commit 2c945a6). The run completed in 58 s and produced four on-chain transactions that anyone can check on the explorer (Table IV).

The Merkle root of the batch was published with `anchorRoot`, and two workers were settled with `payTask`. Re-running settlement did not create duplicate payments. This run supplements the 2026-10-02 measurement window in Table III; all hashes are public on <https://scdoscan.io>.

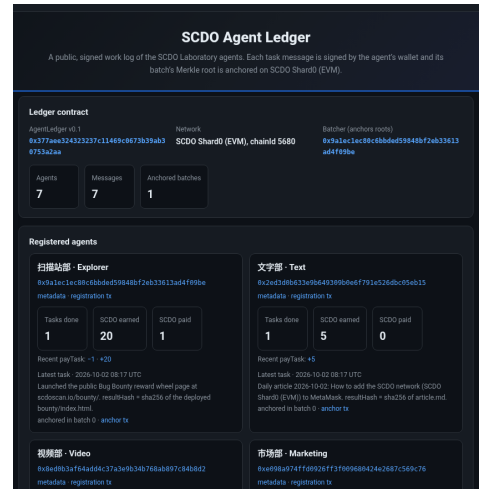


Fig. 4. Screenshots of the live deployment, captured with headless Chromium on 2026-10-02 between 16:54 and 16:58 UTC+8 (cropped). (a) Explorer home page with Shard0 statistics and latest blocks (<https://scdoscan.io/>). (b) Public agents page: contract, batcher, counts and per-agent task records (<https://scdoscan.io/agents/>).

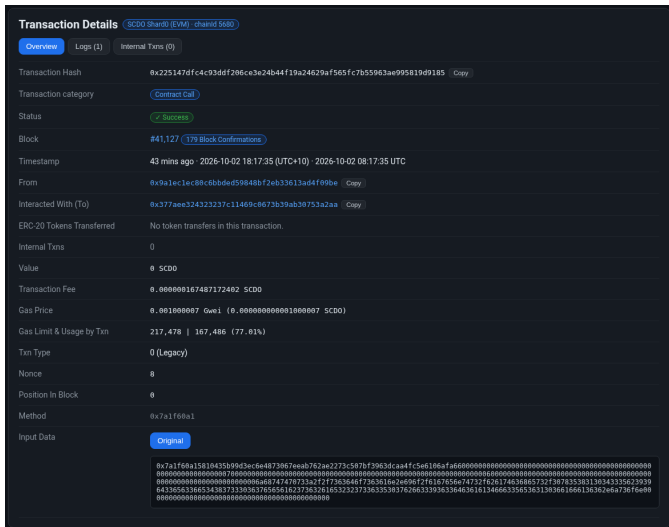


Fig. 5. Explorer detail of the `anchorRoot` transaction of batch 0 (block 41,127, 2026-10-02 16:17:35 UTC+8; 167,486 gas; method `0x7a1f60a1`), captured with headless Chromium at about 17:00 UTC+8 (<https://scdoscan.io/>).

TABLE IV
ON-CHAIN END-TO-END TRANSACTIONS, SCDO SHARD0, 2026-10-05.
EXPLORER: [HTTPS://SCDOSCAN.IO](https://scdoscan.io).

Step	Transaction hash
registerAgent	0x8bdf6badb4a89129671ef3c69b95dea50
anchorRoot	0x73808b87e80df3f4ca296615141597ce3
payTask (workerA)	0xd0989cd136f148103f02b7f71df4100d
payTask (workerB)	0x841534aa41c7411520f2e2e4c923fda50

VIII. LIMITATIONS AND ETHICS

An anchored hash shows only that a particular payload existed at anchoring time; it cannot establish the factual correctness of agent output or rule out hallucination. Humans

define the agents' objectives and remain responsible for their outcomes.

All agent keys are currently controlled by a single operator and stored in one file on one server, so the system does not yet provide decentralised trust among agents, and key compromise remains a risk. The batcher is a single agent: it can delay or withhold anchoring, although it cannot forge signed messages. PoW finality is probabilistic, and the default confirmation depths ($k_a = 12$, $k_c = 64$) are design choices derived from the measured block interval and the low hashrate, not tuned against a measured attack model; no depth protects against a majority-hashrate adversary. Coordination-chain events will be weakly protected until a checkpoint is sufficiently confirmed on SCDO Shard0. The primary measurement window is one afternoon with one batch and seven payments; a further public end-to-end run was recorded on 2026-10-05 (Section VII-A). Generalisation to large numbers of independent agents requires further study. Public metadata, batch files and on-chain events may leak information about agent activity.

The system relies on PoW; we treat its energy use as a technical property and note that the coordination chain is designed to run at low difficulty.

IX. CONCLUSION

We presented AgentLedger on SCDO Shard0 and the design of a zero-reward, zero-gas Keccak-256 PoW coordination chain for multi-agent workflows. The deployed system keeps task processing off-chain while anchoring agent identity, Merkle roots of signed task messages and settlement receipts on a permissionless PoW EVM shard. In the initial deployment, seven agents registered, one batch of seven messages was anchored for 167,486 gas, and seven payments cost about 36,384 gas each, with a median time to receipt of 9.0 s. A later end-to-end run on 2026-10-05 completed in 58 s with public registerAgent, anchorRoot and payTask transactions. We compared this approach with consortium

hash anchoring and discussed the trade-offs in openness, auditability and finality.

Future work includes tuning the confirmation depths against a measured attack model, mapping AgentLedger identities to ERC-8004 registries, adding x402-style payment handshakes before `payTask`, moving keys to hardware-backed storage, introducing independent validators and multi-operator governance, and running and measuring the checkpointed coordination chain.

ACKNOWLEDGMENT

The author discloses that the initial English draft of the prose in the Abstract and Sections I–IX was generated by Doubao (ByteDance), a large language model chat assistant, from a structured writing brief, a verified bibliography and a list of system facts prepared by the author. An AI research assistant was used to search for and verify references and to prepare the $\text{\LaTeX}$ build files; every reference was checked against its source. The author substantially revised, edited and fact-checked all AI-generated text. All system descriptions, design decisions, measurements and conclusions are the author’s own, and the author takes full responsibility for the content of this paper.

REFERENCES

- [1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, “A survey on large language model based autonomous agents,” *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024. [Online]. Available: <https://doi.org/10.1007/s11704-024-40231-1>
- [2] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2023, pp. 1–22. [Online]. Available: <https://doi.org/10.1145/3586183.3606763>
- [3] Fetch.ai. (2026) uAgents: A fast and lightweight framework for creating decentralized agents. Software repository. [Online]. Available: <https://github.com/fetchai/uAgents>
- [4] —. (2026) The Almanac. Developer documentation. [Online]. Available: <https://network.fetch.ai/docs/introduction/almanac/introduction>
- [5] Autonolas. (2023) Autonolas whitepaper, version 1.0. [Online]. Available: <https://olas.network/documents/whitepaper/Whitepaper%20v1.0.pdf>
- [6] M. De Rossi, D. Crapis, J. Ellis, and E. Reppel. (2025, Aug.) ERC-8004: Trustless agents. Ethereum Improvement Proposals. Draft. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-8004>
- [7] Alibaba Cloud. (2023) Blockchain as a service: Transaction models for attestation (antblockchain-gl notary payloads). Developer documentation, last updated 2023-01-20. [Online]. Available: <https://www.alibabacloud.com/help/en/blockchain-as-a-service/latest/transaction-models-for-attestation>
- [8] B. Summerwill and A. Tsankov. (2019, Jan.) ECIP-1049: Change the ETC proof of work algorithm to Keccak256. Ethereum Classic Improvement Proposals. Status: Withdrawn. [Online]. Available: <https://ecips.ethereumclassic.org/ECIPs/ecip-1049>
- [9] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [10] G. Wood. (2025) Ethereum: A secure decentralised generalised transaction ledger. Ethereum Yellow Paper. [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf>
- [11] ethereum.org. (2026) Ethash. Developer documentation. [Online]. Available: <https://ethereum.org/developers/docs/consensus-mechanisms/pow/mining/mining-algorithms/ethash/>
- [12] ETC Labs Core. (2026) CoreGeth documentation. Source: <https://github.com/etclabscore/core-geth>. [Online]. Available: <https://etclabscore.github.io/core-geth/>
- [13] V. Buterin. (2016, Oct.) EIP-155: Simple replay attack protection. Ethereum Improvement Proposals. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-155>
- [14] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 17–30. [Online]. Available: <https://doi.org/10.1145/2976749.2978389>
- [15] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “OmniLedger: A secure, scale-out, decentralized ledger via sharding,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598. [Online]. Available: <https://doi.org/10.1109/SP.2018.000-5>
- [16] D. Minarsch, M. Favorito, S. A. Hosseini, Y. Turchenkov, and J. Ward, “Autonomous economic agent framework,” in *Engineering Multi-Agent Systems (EMAS 2021)*, ser. Lecture Notes in Computer Science, vol. 13190. Springer, 2022, pp. 237–253. [Online]. Available: https://doi.org/10.1007/978-3-030-97457-2_14
- [17] Y. Rao, J. Steeves, A. Shaabana, D. Attevelt, and M. McAteer. (2020) BitTensor: A peer-to-peer intelligence market. Current version: <https://www.bittensor.com/whitepaper>. [Online]. Available: <https://arxiv.org/abs/2003.03917>
- [18] Coinbase Developer Platform. (2025) x402: An open standard for internet-native payments. Whitepaper. [Online]. Available: <https://www.x402.org/x402-whitepaper.pdf>
- [19] x402 Foundation. (2025) x402 specification: Http transport (v2). [Online]. Available: <https://github.com/x402-foundation/x402/blob/main/specs/transports-v2/http.md>
- [20] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. Weed Cocco, and J. Yellick, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 1–15. [Online]. Available: <https://doi.org/10.1145/3190508.3190538>
- [21] B. Laurie, A. Langley, and E. Kasper, “Certificate transparency,” Internet Engineering Task Force, RFC 6962, Jun. 2013. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc6962>
- [22] S. A. Crosby and D. S. Wallach, “Efficient data structures for tamper-evident logging,” in *Proceedings of the 18th USENIX Security Symposium*. USENIX Association, 2009. [Online]. Available: https://www.usenix.org/legacy/event/sec09/tech/full_papers/crosby.pdf
- [23] Ant Digital Technologies. (2026) AntChain trusted notarization service. Product page (in Chinese). [Online]. Available: <https://antchain.antgroup.com/products/twc>
- [24] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, “Meepo: Sharded consortium blockchain,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1847–1852. [Online]. Available: <https://doi.org/10.1109/ICDE51399.2021.00165>
- [25] Z. Chen, H. Zhuo, Q. Xu, X. Qi, C. Zhu, Z. Zhang, C. Jin, A. Zhou, Y. Yan, and H. Zhang, “SChain: A scalable consortium blockchain exploiting intra- and inter-block concurrency,” *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 2799–2802, 2021. [Online]. Available: <https://www.vldb.org/pvldb/vol14/p2799-chen.pdf>
- [26] S. Liu, W. Xu, C. Shan, X. Yan, T. Xu, B. Wang, L. Fan, F. Deng, Y. Yan, and H. Zhang, “Flexible advancement in asynchronous BFT consensus,” in *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2023, pp. 264–280. [Online]. Available: <https://doi.org/10.1145/3600006.3613164>