PATENT APPLICATION

INVENTOR(S)

[0001] KIMZEY, Samuel C

TITLE

[0002] Stateless Action Engine & Minimal Proof Logging

TECHNICAL FIELD

[0003] The invention relates to execution models and audit logging in distributed ledger systems, and more particularly to stateless action engines that execute transformations without storing intermediate state, while producing cryptographic proof artifacts sufficient for full deterministic replay.

BACKGROUND

[0004] Conventional blockchain platforms store complete state transitions or rely on external databases to track evolving system state, resulting in storage overhead, audit complexity, and potential privacy leaks. There is a need for a minimalist execution framework in which actions can be run statelessly, proofs are minimal yet sufficient, and full state can be reconstructed on demand without requiring permanent storage of all intermediate state.

SUMMARY

[0005] The Stateless Action Engine & Minimal Proof Logging framework comprises:

1. Defining actions as input-output transformations with no retained memory or context.
2. Verifying input integrity using cryptographic proofs or attestations.

3. Executing actions deterministically in a stateless engine.
4. Producing a compact proof that summarizes input, output, and action metadata.
5. Logging only this proof to the chain via one-action-one-mint.
6. Reconstructing full application state at any time by replaying proofs through the engine.

DETAILED DESCRIPTION

[0006] Each action is an atomic transformation with a defined input/output schema. When an action is invoked, the system verifies input data integrity using hash commitments or zero-knowledge proofs, then executes the transformation logic in a stateless environment with no persistent memory between actions. No working memory or session state is persisted between actions.

[0007] After execution, a cryptographic proof is generated containing a hash of the inputs, outputs, and a canonical action identifier. This proof is minted on-chain via a one-action-one-mint entry. No internal state is retained on-chain beyond the proof logs.

[0008] Full system state—such as account balances, ledger positions, or task completions—can later be reconstructed by replaying the logged proofs in sequence through the same stateless engine.

METHOD FLOW

1. Action Definition – Define actions with input/output schemas and computation logic.
2. Proof Verification – For each action invocation, verify input proofs or data integrity.
3. Stateless Execution – Execute the action logic in a stateless environment.
4. Proof Generation – Generate a cryptographic proof summarizing the action inputs and outputs.
5. Proof Logging – Mint a one-action-one-mint event recording the proof on-chain.
6. State Reconstruction – Replay logged proofs through the engine to reconstruct full state.

NARRATIVE WORKED EXAMPLE

[0009] A user transfers 50 units to another. The engine verifies the sender's balance proof, executes the transfer statelessly, and creates a proof containing hashes of sender, receiver, amount, and transfer ID. This is minted as a single on-chain entry. To verify system state later, an auditor replays the sequence of transfer proofs and reconstructs balances without any other persistent data.

ALGORITHMIC WORKED EXAMPLE

[00010] Pseudocode:

```
for proof_event in fetchProofEvents():
    action_id, input_hash, output_hash = parseProof(proof_event)
    input_data = retrieveByHash(input_hash)
    computed_output = executeAction(action_id, input_data)
    assert hash(computed_output) == output_hash
    updateStateCache(action_id, computed_output)
    log_audit_trace(action_id, input_hash, output_hash)
```

POTENTIAL EMBODIMENTS

[00011] Use of zero-knowledge proofs to conceal input/output while still proving transformation integrity.
[00012] Hierarchical workflows where complex processes are composed of nested stateless action steps.

[00013] Off-chain proof aggregators that batch multiple proofs into a single mintable commitment.
[00014] Deployment within WASM or VM environments embedded in on-chain systems.
[00015] Governance-controlled updates to allowable actions and proof schemas.

IMPLEMENTATION NOTES

[00016] Action logs follow the one-action-one-mint model. The engine itself may reside in on-chain protocol logic, embedded WASM modules, or pressure-triggered runtime environments. The system never requires global state storage—only the proof log is persistent.

CLAIMS

1. A method for executing and logging actions statelessly in a decentralized system, comprising:
    a. defining, by a processor, an action with fixed input and output schemas;
    b. verifying, by the processor, cryptographic integrity of the action inputs;
    c. executing, by the processor, the action logic without retaining memory state;
    d. generating, by the processor, a cryptographic proof of the action transformation;
    e. recording, by the processor, the cryptographic proof as a one-action-one-mint entry on-chain.
2. The method of claim 1, further comprising reconstructing full system state by replaying the logged proofs through the stateless engine.
3. The method of claim 1, wherein the cryptographic proof comprises hashes of input data, output data, and action identifiers.
4. The method of claim 1, wherein proof events are aggregated off-chain before minting as a combined event.

5. The method of claim 1, wherein the cryptographic proof is a zero-knowledge proof attesting to the correctness of the transformation without revealing underlying data.
6. The method of claim 1, wherein the stateless engine is implemented as a WASM runtime or virtual machine module embedded within the decentralized system's execution environment.
7. The method of claim 1, wherein actions are triggered by reflex conditions derived from memory pressure or flow signals.

ABSTRACT

[00017] A method for executing stateless transformations in a decentralized system, where each action produces a minimal proof of its inputs and outputs. Only these proofs are logged on-chain, enabling full system state reconstruction via deterministic replay without persistent state storage.