Stateless Action Engine & Minimal Proof Logging

1. Abstract:

A method for executing arbitrary action sequences in a stateless execution environment within a decentralized memory-driven system, where only minimal cryptographic proofs of action inputs and outputs are logged on-chain, enabling full state reconstruction without persistent state storage.

2. Technical Field:

The present invention relates to execution models and audit logging in distributed ledger systems, and more particularly to stateless action execution engines that record minimal proof artifacts.

3. Background:

Conventional blockchain and application platforms store complete state changes on-chain or in off-chain databases, incurring high storage costs and privacy concerns. There is a need for a lightweight execution framework that allows arbitrary actions to be performed statelessly while retaining the ability to audit and reconstruct state on demand.

4. Summary:

Defining action specifications as input-output transformations without persisting intermediate state.

Generating cryptographic proofs for each action's inputs and outputs.

Executing actions in a stateless engine that consumes proofs as inputs.

Logging only minimal proof artifacts as one-action-one-mint events on-chain.

Reconstructing full application state by replaying logged proofs through the engine.

5. Detailed Description:

At the conceptual level, this framework treats each action as an atomic, stateless computation. Actions are defined by clear input and output schemas. When an action is invoked, the engine verifies required proofs for inputs, executes the computation deterministically, and produces outputs. A compact cryptographic proof capturing the input hash, output hash, and action identifier is then minted on-chain. Full system state can later be reconstructed by replaying the sequence of proof artifacts through the same stateless engine.

6. Method Flow:

Step 1: Action Definition – Define actions with input/output schemas and computation logic.

Step 2: Proof Verification – For each action invocation, verify input proofs or data integrity.

Step 3: Stateless Execution – Execute the action logic in a stateless environment.

Step 4: Proof Generation – Generate a cryptographic proof summarizing the action inputs and outputs.

Step 5: Proof Logging – Mint a one-action-one-mint event recording the proof on-chain.

Step 6: State Reconstruction – Replay logged proofs through the engine to reconstruct full state.

7. Narrative Worked Example:

Consider a payment action: a user submits a transfer of 50 units to another account. The engine verifies the user's balance proof, executes the transfer statelessly, and produces a proof containing hashes of the sender, receiver, and amount. This proof is minted on-chain. To audit account balances, a verifier replays all transfer proofs in order, recomputing balances without any additional state storage.

8. Algorithmic Worked Example:

Pseudocode:

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

9. Potential Embodiments:

Using zero-knowledge proofs for enhanced privacy of action data while logging minimal commitments.

Hierarchical action engines where complex workflows decompose into nested stateless actions.

Off-chain proof aggregators that batch multiple proofs into combined mint events for efficiency.

Integration with on-chain VMs or main application modules to natively support stateless actions.

Dynamic action definition updates via governance-managed proof schemas.

10. Implementation Notes:

Action proofs follow the one-action-one-mint paradigm and are stored immutably on-chain. No persistent on-chain state is maintained beyond the proofs; the engine's code resides in protocol modules or VM contexts.

11. Claims:

1. A method for executing and logging actions statelessly in a memory-driven blockchain system, comprising:

a. defining, by a processor, an action with specified input and output schemas;

b. verifying, by the processor, cryptographic proofs for the action inputs;

c. executing, by the processor, the action logic in a stateless environment;

d. generating, by the processor, a cryptographic proof capturing the action inputs, outputs, and identifier;

e. recording, by the processor, the cryptographic proof as a one-action-one-mint event on-chain;

2. The method of claim 1, further comprising reconstructing system state by replaying the logged proofs through the stateless execution 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 or batched off-chain before minting combined events.

5. The method of claim 1, wherein zero-knowledge proofs are used to attest action correctness without revealing underlying data.