Selective Memory Disclosure via Zero-Knowledge Proofs

1. Abstract:

A method for enabling selective disclosure of information in decentralized memory logs by generating zero-knowledge proofs for specified predicates, allowing verification of memory-derived statements without revealing underlying event data.

2. Technical Field:

This invention relates to privacy-enhancing technologies in blockchain systems, and more particularly to methods for proving properties of historical memory records using zero-knowledge proofs.

3. Background:

On-chain memory logs may contain sensitive information that users wish to keep confidential. Conventional proof methods require revealing data to prove statements, compromising privacy. Zero-knowledge proofs (ZKPs) offer privacy-preserving verification but have not been applied to selective disclosure in memory-driven systems.

4. Summary:

Disclosed is a Selective Memory Disclosure mechanism comprising:

1. Selecting one or more predicates about memory logs (e.g., existence, range, order).

2. Constructing a zero-knowledge proof circuit for the predicates.

3. Generating a cryptographic proof without revealing underlying data.

4. Verifying the proof on-chain or off-chain to confirm predicates hold.

5. Optionally triggering downstream memory-flow or value-flow processes upon verification.

5. Detailed Description:

The method involves retrieving relevant memory entries from decentralized sources, encoding desired predicates into ZKP circuits (e.g., SNARKs, STARKs, or Bulletproofs), and executing proof generation to produce witness-based proofs. Proofs and minimal public inputs are then submitted for verification, ensuring predicates such as event existence, value ranges, or temporal relationships hold without exposing raw logs.

6. Method Flow:

Step 1: Predicate Selection – Identify predicates about memory logs to prove (e.g., event inclusion, tension range, chronological order).

Step 2: Circuit Construction – Translate predicates into an arithmetic or Boolean circuit suitable for a ZKP system.

Step 3: Proof Generation – Compute the circuit witness from actual memory log data and generate a zero-knowledge proof.

Step 4: Proof Submission – Provide the proof and public inputs to a verifier component, which may be a smart contract or off-chain verifier.

Step 5: Verification – Confirm the proof against a verification key, validating the predicates without revealing the underlying logs.

7. Narrative Worked Example:

A participant wishes to prove that a transaction 'TX42' occurred before block height 500 without revealing transaction details. They define the predicate 'TX42 exists at index ≤ 500'. A SNARK circuit embeds Merkle path verification and index bound checks. The user generates the proof and submits it to a smart contract. Verifiers confirm the proof, validating existence without revealing transaction data or amounts.

8. Algorithmic Worked Example:

Using a SNARK system, define Public Inputs: Root, IndexBound. Witness Inputs: PathElements, PathIndices, ActualIndex. Circuit enforces: HashChain(PathElements, PathIndices) == Root AND ActualIndex ≤ IndexBound. Pseudocode:

1. path = get_merkle_proof('TX42')
2. circuit = compile_zkp_circuit(path.schema)
3. witness = { 'elements': path.elements, 'indices': path.indices, 'index': path.index }
4. proof = SNARK.prove(circuit, witness, public_inputs={ 'root': path.root, 'index_bound': 500 })
5. valid = SNARK.verify(circuit.vk, proof, public_inputs={ 'root': path.root, 'index_bound': 500 })

9. Potential Embodiments:

Range Proofs – Prove that memory-tension metrics are within a confidential interval.

Order Proofs – Demonstrate that event A preceded event B without revealing timestamps.

Composite Predicates – Combine multiple conditions (e.g., existence AND range) in a single proof.

On-Chain Verification – Deploy proof verification logic in smart contracts for fully on-chain validation.

Flow Trigger – Trigger memory-flow or surplus-flow processes upon proof success.

10. Implementation Notes:

Proof circuits and verification keys may be stored or referenced via on-chain transactions.

Proof generation and verification follow a one-action, one-mint model for auditable trail.

No raw memory logs or sensitive details are disclosed during the proof lifecycle.

Verifiers can be decentralized oracles, smart contracts, or off-chain services.

11. Claims:

1. A method for selectively disclosing properties of decentralized memory logs without revealing underlying data, the method comprising:

a. identifying one or more predicates describing properties of memory logs stored on a blockchain;

b. constructing a zero-knowledge proof circuit encoding the predicates over memory log data;

c. generating a zero-knowledge proof for the predicates without exposing the memory log data;

d. providing the proof and verification key to a verifier; and

e. verifying the proof to confirm the predicates hold without revealing underlying log entries.

2. The method of claim 1, wherein the predicates include existence proofs for specific events.

3. The method of claim 1, wherein the predicates include range proofs over tension metrics.

4. The method of claim 1, wherein the predicates include chronological ordering proofs.

5. The method of claim 1, wherein verification of the proof triggers downstream processes.