PATENT APPLICATION

INVENTOR(S)

[0001] KIMZEY, Samuel C

TITLE

[0002] Selective Memory Disclosure via Zero-Knowledge Proofs

TECHNICAL FIELD

[0003] 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.

BACKGROUND

[0004] 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.

SUMMARY

[0005] Disclosed is a Selective Memory Disclosure mechanism that enables verifiable assertions over private memory logs without exposing underlying content. The system supports existence, range, and order predicates via zero-knowledge proof circuits. Proofs are generated using memory data, minted as standalone attestations, and verified on- or off-chain. This enables downstream processing, such as value flow or trait activation, based solely on confirmed properties, while preserving the confidentiality of the memory origin.

DETAILED DESCRIPTION

[0006] 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.

[0007] Zero-knowledge proof systems include a variety of cryptographic protocols that enable verification of statements without revealing the underlying data. Common types include:

1. SNARKs (Succinct Non-interactive Arguments of Knowledge): compact, fast-to-verify proofs ideal for smart contract environments.
2. STARKs (Scalable Transparent Arguments of Knowledge): transparent, post-quantum resistant alternatives that avoid trusted setup.
3. Bulletproofs: shorter proofs for range-type statements with no trusted setup, used in confidentiality-preserving systems.

METHOD FLOW

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

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

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

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

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

NARRATIVE WORKED EXAMPLE

[0008] 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.

ALGORITHMIC WORKED EXAMPLE

[0009] A participant wishes to prove the predicate: "TX42 exists before index 500" without disclosing transaction details.

1. Public Inputs:

    root: Merkle root of the memory log

    index_bound: upper bound of acceptable index (e.g., 500)

2. Private Witness (inputs):

   path.elements: Merkle proof path elements

   path.indices: Merkle sibling directions

   path.index: actual index of TX42 in log

3. Circuit Logic:

```
    Enforce: verify_merkle_path(path.elements, path.indices) == root
    Enforce: path.index ≤ index_bound
```

[00010] Pseudocode:

```
path = get_merkle_proof('TX42')
circuit = compile_zkp_circuit('exists_before_index')
witness = {
   'elements': path.elements,
   'indices': path.indices,
   'index': path.index
}
public_inputs = {
   'root': path.root,
   'index_bound': 500
}
proof = SNARK.prove(circuit, witness, public_inputs)
valid = SNARK.verify(circuit.vk, proof, public_inputs)
```

POTENTIAL EMBODIMENTS

[00011] Range Proofs – Prove that memory-tension metrics are within a confidential interval.

[00012] Order Proofs – Demonstrate that event A preceded event B without revealing timestamps.

[00013] Composite Predicates – Combine multiple conditions (e.g., existence AND range) in a single proof.

[00014] On-Chain Verification – Deploy proof verification logic in smart contracts for fully on-chain validation.

[00015] Flow Trigger – Trigger memory-flow or surplus-flow processes upon proof success.

IMPLEMENTATION NOTES

[00016] Zero-knowledge proof circuits (e.g., SNARKs, STARKs, Bulletproofs) are constructed per predicate and stored or referenced via on-chain identifiers.

[00017] Each proof generation follows a one-action-one-mint model, ensuring every selective disclosure event is individually auditable.

[00018] No raw memory logs are disclosed; all predicates operate on cryptographic commitments or Merkle roots.

[00019] Verifiers may include smart contracts, trait modules, or off-chain oracles, depending on flow requirements.

[00020] Predicate logic may optionally be embedded in Trait circuits for reflexive memory evaluation.

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 bounded or confidential numeric values such as memory tension or flow metrics.
4. The method of claim 1, wherein the predicates include chronological ordering proofs verifying temporal or positional sequence of memory events.
5. The method of claim 1, wherein verification of the proof triggers downstream processes.
6. The method of claim 1, further comprising minting the zero-knowledge proof as an on-chain event, forming an immutable record of the selective disclosure.
7. The method of claim 1, wherein the verifier is a decentralized oracle network, autonomous agent, or smart contract.
8. The method of claim 1, wherein successful proof verification triggers a memory-dependent system process, including resource allocation, agent spawning, or value redistribution.
9. The method of claim 1, wherein the predicate is initiated or defined by a Trait module in response to observed system conditions.

ABSTRACT

[00021] A method for selectively disclosing properties of on-chain memory logs using zero-knowledge proofs. Predicates such as existence, range, or order are encoded into circuits and verified without revealing raw data. Each proof forms a standalone mintable event, enabling downstream execution or flow activation while preserving memory privacy and auditability.