Document 5: Relational Merkle Closures

---

1. Abstract

Relational Merkle Closures provide cryptographic proof that **no valid dependency path exists** between assets within a directed graph or dataset at a specific epoch. For each source asset and epoch, the system computes its full set of reachable nodes (closure), encodes that set as a Sparse Merkle Map (SMM), and commits its root into an authenticated, chained accumulator. A **non-reachability receipt** then proves that a target asset B is *not* reachable from asset A by presenting:

- a Sparse Merkle non-membership proof for B under A's closure root,
- inclusion of that closure root in the epoch accumulator,
- and anchoring of that accumulator in a cumulative root-chain.
  This construction enables privacy-preserving proof of absence — without revealing intermediate nodes, graph structure, or asset content.

2. Technical Field

This document operates within:

- Authenticated data structures for graphs and relational systems
- Cryptographic proofs of *absence* (non-reachability, non-membership)
- Sparse Merkle trees, accumulators, and anchor-based audit trails
- Supply chain integrity, dependency validation, SBOMs, access control, lifecycle proofs

---

3. Background and Problem

Linear Merkle trees prove *membership*. They cannot answer:
"Can asset A reach asset B?"
"Is there any valid path between two nodes?"

Relational systems (supply chains, access-control graphs, dependency DAGs, key hierarchies) require **proofs of non-reachability**, i.e., $A \not\Rightarrow B$. Traditional systems:

- require full graph disclosure, or
- rely on trusted assertions instead of cryptographic evidence.

Relational Merkle Closures solve this by:

- committing to entire reachability sets per asset, per epoch,
- and allowing compact Merkle *non-membership* proofs to show absence of a connection.

4. Key Definitions

Term	Meaning
Source Node (A)	Node whose reachability closure is computed
Target Node (B)	Node tested for reachability from A
Closure (Desc_i(A))	Set of all nodes reachable from A at epoch *i*
Sparse Merkle Map (SMM)	Key → value map with default for non-members
Closure Root (SMT_root_A^i)	Root hash of A's closure SMM at epoch *i*
Closure Header (CH_A^i)	Hash binding {A, epoch i, closure root}
Epoch Accumulator	Merkle/accumulator structure over closure headers
Root Chain	Cumulative chain of epoch roots to enforce immutability
Non-Reachability Receipt	Proof binding A ↛ B at epoch i

5. System Overview

Core Components:

- **Graph Store (100):** Maintains asset nodes and dependency edges
- **Closure Engine (110):** Computes reachability set Desc_i(A)
- **DAG Validator (120):** Rejects cycles before closure is computed
- **Sparse Merkle Map Builder (130):** Encodes closure as SMM → generates SMT_root_A^i
- **Closure Header (150):** Binds source, epoch, and closure root
- **Epoch Accumulator (160):** Includes CH_A^i into per-epoch authenticated structure
- **Root Chain (170):** $Chain_k = H(Chain_{k-1} \| Epoch\_Root_k)$
- **Anchor Interface (180):** Publishes roots externally (optional)
- **Receipt Store (190):** Stores non-reachability receipts

(Figure references: See Diagrams 1–4 in uploaded source)

6. Core Mechanism

6.1 Closure Construction

For each source A at epoch i:

Desc_i(A) = { all nodes reachable via valid edges }

- Computed via deterministic DFS/BFS
- Requires graph to be acyclic for epoch i

6.2 Sparse Merkle Encoding

- Domain: Universal keyspace of all node IDs
- Reachable nodes → mapped to 1 or asset-hash
- Unreachable nodes → map to canonical default ⊥
- Resulting root: $SMT\_root\_A^i$

6.3 Closure Header

$CH\_A^i = H("closure" \| A \| i \| SMT\_root\_A^i)$

Inserted into Epoch Accumulator → Root Chain.

---

7. Non-Reachability Receipt ($A \not\Rightarrow B$)

Receipt contains:

Field	Contents
Source Asset	A_id
Target Asset	B_id
Epoch	i
Closure Header	CH_A^i
Closure Root	$SMT\_root\_A^i$
Sparse Merkle Proof	$\pi\bot$ such that $H(B) = \bot$ under $SMT\_root\_A^i$
Epoch Accumulator Proof	π_acc confirming CH_A^i inclusion
Root Chain Proof	Confirms CH_A^i is anchored chronologically
Optional	Valid-edge root, tombstone proofs for revoked edges

8. Edge Validity Models

Two supported models (in spec & claims):

Model	Description
Dual-Set (Valid + Invalid)	Two Merkle roots per epoch: R_v^i (valid edges), R_i^i (invalid/tombstones)
Interval-Based	Each edge has start_epoch, end_epoch; valid only if epoch satisfies range

Both allow:

- Proof of valid dependency

- Proof of revoked or removed edge
- Proof of "dangling" edges

---

9. Verification Procedure

To verify $A \not\Rightarrow B$ at epoch i:

1. Validate CH_A^i commits to $(A, i, SMT\_root\_A^i)$.
2. Confirm $\pi\bot \rightarrow B$ maps to default = $\bot$ under $SMT\_root\_A^i$.
3. Confirm $CH\_A^i \in$ epoch accumulator via π_acc.
4. Confirm epoch accumulator $\in$ root chain.
5. (Optional) Validate edge validity or tombstones.

If all hold → No valid path exists from A to B at epoch i.

---

10. Example Applications

- Supply chain: prove Supplier X does *not* lead to Component Y.
- SBOM / software dependencies: prove Package A does *not* include Library B.
- Access control: user cannot reach resource.
- Revocation: prove that a dependent asset is no longer reachable once upstream is destroyed.
- Database integrity: show that foreign key paths do not exist.

---

11. Advantages

✔ First cryptographic proof model for *absence of dependency*
✔ Scales logarithmically with graph size (Sparse Merkle)
✔ No disclosure of graph topology or intermediate nodes
✔ Compatible with Documents 1–4 (EIV, Transition States, CSTP Core, Graph Lifecycle)
✔ Supports destruction, migration, revocation, and non-existence universally

---

End of Document 5 – Relational Merkle Closures