Document 4: CSTP – Graph Lifecycle

---

1. Abstract

The Graph Lifecycle Model extends Cryptographic State Transition Proofs (CSTP) from linear, single-asset timelines to multi-asset, branching, merging, and interdependent state graphs. Each asset is represented as a node, and each transition is represented as a cryptographically bound edge. The system ensures that at any point in time, an auditor can prove not only that a given transition occurred, but also that it was consistent with all related assets, constraints, and prior states within the graph — without accessing any of the underlying data.

---

2. Technical Field

This work operates within:

- Distributed cryptographic state management
- Directed acyclic graphs (DAGs) for provenance and lifecycle
- Multi-asset and multi-epoch state validation
- Irreversible cryptographic auditability and dependency tracking

---

3. Background

Linear state models (Document 2) solve single-asset state transition proofs. However, real systems require:

- **Assets that branch** (forking a dataset into derivatives)
- **Assets that merge** (combining multiple records or keys)
- **Assets that depend on others** (compliance chains, root-of-trust structures)
- **Cross-asset destruction or expiry**
- **Simultaneous transitions across multiple nodes**

Existing Merkle-based systems track independent logs or chains, but do not encode interdependencies. This document defines a general model: **Merkle-anchored transition DAGs**.

---

4. Definitions

Term / Symbol	Meaning
Node (N_i)	Represents an asset or sub-asset in the system
Edge (E_{ij})	A transition from node i to node j
Transition-State (TS)	Cryptographic record of an edge (reuse from Document 2)
DAG	Directed Acyclic Graph of transitions; cycles are disallowed
Parent Set $P(N_i)$	All nodes whose outputs feed into N_i
Child Set $C(N_i)$	All nodes derived from N_i
Asset Graph (G)	Set of all N and E defining an ecosystem of states
Merge Transition	Multiple TS inputs → one output
Fork Transition	One TS input → multiple outputs

5. System Overview

Core Components (extends Document 3 CSTP Core):

- **Graph Controller** – Maintains valid DAG of nodes and transitions
- **Dependency Validator** – Ensures no cycles or invalid parent references
- **Graph Merkle Accumulator** – Commits all edges (transitions) into per-epoch Merkle trees
- **Root Chain** – Chained hash of epoch roots (same as Document 2 & 3)
- **Graph Receipt Store** – Stores each transition-state receipt along with parent references
- **Verifier** – Independently reconstructs graph branches to confirm transitions

---

6. Core Mechanism

6.1 Graph Construction

- Each transition (creation, migration, merge, destruction) becomes an **edge E_i**
- Each asset or sub-asset becomes a **node N_i**
- Edges always move forward; no backward cycles permitted
- Each edge includes:
    - asset_id or multiple IDs (merge case)
    - parent_commitment(s)
    - commitment_current

- epoch, transition_type

6.2 Merge and Fork Logic

Transition Type	Definition
Fork	$N_i \to \{N_j, N_k\}$. E.g. dataset duplicated or branched
Merge	$\{N_a, N_\beta\} \to N_\gamma$. E.g. key shares recombined, datasets joined
Multi-Destroy	$\{N_a, N_\beta\} \to$ NULL digest states
Migration	N_i (RSA) $\to N_i$ (PQ crypto)

Each transition emits a **transition-state receipt** including all parent references.

6.3 Transition-State Commitment (Graph-Aware)

```
C_graph = H(
   sorted(asset_ids) ||
   sorted(parent_commitments) ||
   epoch ||
   transition_type ||
   salt
)
```

6.4 Merkle and Chain Anchoring

- All C_graph values for epoch e → Merkle root R_e
- Root chain persists: $chain_e = H(chain_{e-1} \| R_e)$
- Verifiable without asset data, only receipts + commitments

---

7. Receipt Structure (Graph Version)

Field	Description
asset_ids	One or more identifiers involved in transition
parent_commitments	Commitments of all input states
commitment	New transition commitment
merkle_path	Proof of inclusion in epoch Merkle tree
epoch_root	Merkle root of that epoch
chain_root	Cumulative chain of roots
state_before	Optional hash of combined inputs
state_after	Hash of output or NULL_DIGEST for destruction

8. Verification

To verify a transition:

1. Check no cycles exist in parent chain
2. Recompute C_graph from asset_ids + parent_commitments
3. Reconstruct Merkle path → confirm matches epoch_root
4. Confirm epoch_root belongs in root chain
5. Optionally verify state_before/state_after matches expectations (if provided)
6. Accept transition only if chain continuity is intact

9. Example Graph Scenarios

9.1 Fork Example

Dataset A (N1)
  |
  → TS1: N1 → N2 (subset)
  → TS2: N1 → N3 (redacted version)

9.2 Merge Example

Key Share K1 + Key Share K2 → Recombined Key (N4)

9.3 Destruction Cascade

N1, N2, N3 → TS(destroy_all) → NULL Digest states

10. Advantages

✔ Models real-world multi-asset dependencies
✔ Enables proving correct merge/split/migration without accessing data
✔ Works offline — only commitments and receipts needed
✔ Compatible with Documents 1–3 (EIVs, Transition-States, CSTP Core)
✔ Deterministic, acyclic, verifiable structure

End of Document 4 – CSTP Graph Lifecycle