Document 2: Transition-State Model (CSTP Implementation)

---

1. Abstract

This document defines the Transition-State Model, a core component of Cryptographic State Transition Proofs (CSTP). Unlike traditional systems that treat deletions, migrations, or key expirations as operational events or log entries, this model represents each lifecycle change as a persistent, verifiable cryptographic object called a **transition-state**. Each transition-state binds an asset identifier, epoch, and transition type to a commitment, is anchored in a Merkle accumulator, chained against previous transitions, and remains independently verifiable without access to the asset itself. This document builds on Document 1 (EIV Destruction Receipts) but generalizes beyond destruction to all one-way asset state changes.

---

2. Technical Field

This model operates within:

- Cryptographic lifecycle management
- Tamper-evident distributed audit structures
- Asset governance: creation, migration, retention, destruction
- Forward-only state transition systems (CSTP architecture)

---

3. Background and Problem Statement

Most systems today:

- Track only **asset states**, not **transitions**
- Rely on logs or attestations to prove asset deletion or migration
- Cannot cryptographically prove that a transition occurred without revealing underlying data
- Treat lifecycle events as metadata rather than immutable cryptographic objects

This model inverts that approach:

- Transitions themselves become **first-class cryptographic states**
- Every one-way lifecycle change yields a **transition-state receipt**
- Auditors verify transitions without needing the asset

---

4. Definitions and Symbols

Symbol / Term	Meaning
Asset (100)	Any digital object, file, key, dataset, or stateful entity
Transition-State (200)	Persistent cryptographic record of a one-way lifecycle change
Commitment (210)	H(asset_id ‖ epoch ‖ transition_type ‖ salt)
Merkle Path (220)	Authentication branch linking commitment to root
Epoch Root (230)	Merkle root for all transition-states in an interval
Root Chain (240)	H(previous_root_chain ‖ epoch_root)
State-Before (260)	Hash of asset before transition
State-After (270)	Hash after transition (or NULL_DIGEST on destruction)

5. System Model

5.1 Parties

- **State Transition Engine (110)** — Detects lifecycle events and constructs transition-states
- **Commitment Constructor (130)** — Computes cryptographic commitments
- **Merkle Accumulator (140)** — Inserts commitments and computes epoch roots
- **Root Chain Logger (150)** — Chains roots to enforce forward-only progression
- **Anchor Interface (160)** — Publishes roots externally (optional)
- **Receipt Store (190)** — Stores transition-state receipts
- **Verifier (120)** — Independently validates transition-state receipts

5.2 Assumptions

- Hash function is collision-resistant
- Assets undergo **one-way transitions only** (no reversal)
- Destroyed/migrated assets are not needed to verify transitions
- Roots or chain values are stored or externally anchored

---

6. Core Mechanism

6.1 Transition-State Concept

For every asset transition (create, migrate, retain, destroy):

Generate transition-state T:
T = {commitment, merkle_path, epoch_root, root_chain, state-before, state-after}

Store T as persistent receipt

6.2 Commitment Construction

commitment (210) = H(asset_id ‖ epoch ‖ transition_type ‖ salt)

This binds:

- What changed
- When it changed
- Which asset changed

6.3 Merkle Accumulation and Epoch Root

- All commitments in a given epoch are added to Merkle accumulator
- Epoch Root (230) = H(all commitments)
- Stored or externally anchored

6.4 Root Chain (Replay Protection)

root_chain[i] = H(root_chain[i-1] ‖ epoch_root[i])

Prevents tampering or replay of stale transition-states.

---

7. Destruction Transition-State

Destruction is a special case:

Field	Value
commitment	H(asset_id ‖ epoch ‖ "destroy")
state-before (260)	H(asset_valid)
state-after (270)	NULL_DIGEST (reserved constant)
meaning	Proves asset once existed, now cryptographically gone

8. Verification Process

An independent verifier:

1. Receives transition-state receipt (200)
2. Recomputes commitment (210)
3. Validates Merkle path (220) against Epoch Root (230)

4. Verifies Root Chain (240) matches published anchor
5. Confirms state-after = NULL_DIGEST (if destruction)
6. Does **not** need access to asset data

---

9. Example Lifecycle Chain

create → retain → migrate → destroy

T0: H(asset, epoch0, create)
T1: H(asset, epoch3, retain)
T2: H(asset, epoch5, migrate_to_quantum_safe)
T3: H(asset, epoch9, destroy), state-after = NULL
Each T_i anchored in Merkle, chained in root-chain

10. Implementation Notes

- Commitment fields may include regulatory basis hashes or policy identifiers
- Accumulators may operate per region / jurisdiction
- Transition-states can exist even if assets are encrypted, off-chain, or deleted
- This model generalizes beyond EIV/key destruction to complete asset lifecycle management