Document 3: CSTP Core

(Cryptographic State Transition Proofs – Core Model)

---

1. Abstract

The CSTP Core defines the foundational cryptographic mechanism for proving that a digital asset has changed from one state to another in a way that is irreversible, auditable, and does not require access to the asset itself. It unifies forward-secure key evolution, commitment construction, Merkle-based accumulation, and chained anchoring. This document generalizes destruction receipts (Document 1) and transition-state receipts (Document 2) into a full lifecycle system — where **the transitions themselves become the audited truth**, and not the assets.

---

2. Technical Field

This system operates within:

- Cryptographic state lifecycle management
- Tamper-evident logging and integrity proofs
- Forward-only key evolution and irreversible transitions
- Independent verification of digital asset states without disclosure of data

---

3. Background and Problem

Traditional systems record asset states but not state transitions. Logs can be deleted, altered, or forged. There is no cryptographic way to prove that an asset:

- moved from *valid* → *expired*,
- migrated from *scheme A* → *scheme B*, or
- was *destroyed and is no longer reconstructable.*

Existing Merkle or blockchain systems prove existence. CSTP proves **state change**, including change into nonexistence.

---

4. Definitions and Symbols

Symbol / Term	Meaning
ERK_i	Epoch Root Key at epoch i
$ERK_{i+1} = H(ERK_i)$	Forward-secure key progression, old keys destroyed
OWK	Object Wrap Key derived from ERK_i + asset_id
Commitment (C)	H(asset_id ‖ salt ‖ epoch ‖ transition_type)
Transition-State	Cryptographic record of a one-way lifecycle change
Receipt	{ commitment, merkle_path, epoch_root, chain_root, state_before, state_after }
Null Digest	Defined digest indicating absence/destruction

5. System Overview

Core Components:

- **Forward-Secure Key Engine:** Maintains ERK_i values per epoch; destroys older ones.
- **Key Derivation Module:** Derives per-asset OWKs and asset encryption keys.
- **Commitment Constructor:** Converts transitions into cryptographic commitments.
- **Merkle Accumulator:** Aggregates commitments into epoch roots.
- **Root Chain Logger:** Chains epoch roots to enforce chronological immutability.
- **Receipt Generator:** Outputs proof of state transition.
- **Verifier:** Third-party process capable of validating any receipt using only public data.

---

6. Core Mechanism

6.1 Forward-Secure Key Evolution

ERK0 = seed
ERKi+1 = H(ERKi)
Destroy ERKi after use → cannot reconstruct previous epochs.

6.2 Object Wrap Key (OWK) Derivation

OWK = KDF(ERKi, asset_id)
Used to encrypt/decrypt the asset-specific encryption key.
Destroyed at expiry.

6.3 Transition Commitment

C = H(asset_id || salt || epoch || transition_type)

Transition types include:

- create
- retain
- migrate
- destroy

6.4 Merkle Insertion

- Insert commitment C into Merkle tree of epoch i
- Compute epoch_root_i

6.5 Root Chain

$\text{chain}_i = H(\text{chain}_{i-1} \,||\, \text{epoch\_root}_i)$

Prevents removal, reordering, or replay of older states.

---

7. Receipt Structure

Field	Description
commitment	H(asset_id || salt || epoch || transition_type)
merkle_path	Hash siblings from leaf→root
epoch_root	Merkle root of all transitions in that epoch
chain_root	Cumulative chain up to this epoch
state_before	Hash of asset in previous state
state_after	Hash of new state, or NULL_DIGEST for destruction
(optional) attestations	TEE report, MPC signatures, or HSM evidence

8. Root Chain Anchoring

Epoch roots or chain roots may be:

- Anchored to a public ledger, transparency log, or time-stamped notary
- Stored privately in regulated archives or distributed logs
- Used by independent auditors to validate entire history without asset access

---

9. Security Properties

✔ Forward-only time progression (cannot restore past states)
✔ Destruction of ERK_i prevents future reconstruction of OWK and asset keys
✔ Commitments bind asset, epoch, and transition irreversibly
✔ Merkle proof + chain root allows independent verification
✔ No plaintext asset data is required to verify transitions
✔ Destruction, migration, or modification cannot be forged retroactively

---

10. Example: Full Asset Lifecycle

T0: Create → C0, root0, chain0
T1: Retain → C1, root1, chain1
T2: Migrate → C2 (e.g., to quantum-safe), chain2
T3: Destroy → C3, state_after = NULL_DIGEST, chain3

Each step is independent, verifiable, and cryptographically chained.

End of Document 3 – CSTP Core