

EFM Codex — Appendix A

Capsule Integrity and Forensic State Serialization

Internal Auditability and Control Loop Tracing

Entropica SPC — Yology Research Division

Version 1.4 — December 2025

Volume Dependencies

This appendix assumes familiarity with:

- **Volume I** — Capsule definition (§2), Reflex Engine (§3), ΔS computation, ZK-SP
- **Volume II** — Arbiter Layer (§2), d-CTM (§2.7), Probation Protocol (§2.8), Gardener Override (§2.10)

Contents

1 Overview and Purpose	2
1.1 Bridging Summary	2
1.2 Architectural Position	2
2 Formal Definitions	2
3 Trigger Conditions	3
4 State Serialization Format	3
4.1 Canonical Schema	3
4.2 Storage Pipeline	4
4.3 Degraded Mode and Storage Failures	5
4.4 Snapshot Rate Guidelines	5
5 Recovery and Rollback	5
5.1 Rollback Mechanism	5
5.2 Constraints	7
5.3 Replay vs. Rollback	7
5.4 Integration with Provenance Stack (Appendix M)	8
6 Reference Implementation	8
7 Testing and Validation	10
7.1 Test Objectives	10
7.2 Metrics	10
8 Cross-References	10

1 Overview and Purpose

1.1 Bridging Summary

Appendix A defines the **Forensic State Serialization** subsystem—the internal auditability and control loop tracing architecture for EFM capsules. While the Reflex Engine (Vol. I §3) and Arbiter Layer (Vol. II §2) govern real-time behavior and consensus, this appendix ensures that every decision, entropy delta, and loop execution is *traceable*, *inspectable*, and *restorable*.

Key Distinction: This is not passive “post-mortem forensics.” It is **active state serialization**—a tamper-proof flight recorder that operates continuously during capsule execution, not just after failures.

1.2 Architectural Position

Forensic State Serialization spans multiple layers:

Layer	Component	Forensic Role
Layer 0.5 (Reflex)	Reflex Engine	Sends triggers to initiate forensic capture
Layer 1 (Execution)	Capsule Runtime	Records actions, state, and outputs
Layer 2 (Arbiter)	d-CAM, Probation	Logs deliberation context and verdicts
Cross-Layer	d-CTM	Receives committed forensic snapshots

Table 1: Forensic State Serialization layer integration.

2 Formal Definitions

Definition 2.1 (Control Loop Trace). A Control Loop Trace T_c for capsule C at cycle n is a tuple:

$$T_c(n) = (\text{capsule_id}, n, I_n, S_n, O_n, \Delta S_n, ts_n) \quad (1)$$

where:

- I_n = input vector at cycle n
- S_n = internal state snapshot
- O_n = output vector
- ΔS_n = behavioral entropy (Vol. I Definition 3.1)
- ts_n = timestamp

Definition 2.2 (Forensic Snapshot). A Forensic Snapshot F is a signed, ZK-SP anchored Control Loop Trace:

$$F = (T_c, \text{trigger_type}, \text{zkp_hash}, \text{dctm_ref}) \quad (2)$$

where $\text{trigger_type} \in \{\text{REFLEX}, \text{PROBATION}, \text{GARDENER}, \text{EPOCH}\}$.

Definition 2.3 (Trace Ring Buffer). Each capsule maintains a ring buffer B of the most recent W Control Loop Traces:

$$B = [T_c(n - W + 1), \dots, T_c(n)] \quad (3)$$

Default: $W = 1000$ cycles. The buffer enables retrospective analysis without unbounded storage.

3 Trigger Conditions

Forensic snapshot capture is triggered by four conditions:

Trigger Type	Condition	Reference
REFLEX	$\Delta S \geq \tau$	Vol. I §3 (Reflex Engine escalation)
PROBATION	Capsule in Probation state	Vol. II §2.8 (every tick snapshotted)
GARDENER	Gardener Override issued	Vol. II §2.10 (mandatory audit entry)
EPOCH	$n \bmod E = 0$	Routine archival sampling (default $E = 10000$)

Table 2: Forensic snapshot trigger conditions.

Probation Integration (Vol. II §2.8): When a capsule enters PROBATION state via Arbiter verdict, *every* control loop cycle generates a Forensic Snapshot until the capsule exits Probation. This provides dense forensic coverage during high-risk periods.

Gardener Override Log (Vol. II §2.10): The legacy “Manual Request” trigger is replaced by GARDENER. Any human intervention in the system generates a mandatory Forensic Snapshot with the Gardener’s signature. This satisfies ethical constraint E4 (Audit Accessibility, Vol. II §3.9.5).

Cryptographic Consent Required

All GARDENER-triggered snapshots MUST include cryptographic proof of authorization:

- **Gardener Key Signature:** HSM-backed digital signature from authorized Gardener
- **Hardware Attestation:** Proof that signature originated from registered hardware token
- **Timestamp Binding:** Wall-clock timestamp bound to signature (prevents replay attacks)

Anonymous or unsigned manual requests are **rejected**. This prevents DOS attacks via flooding manual trigger requests. See Appendix F §5.1 for Gardener Key management.

4 State Serialization Format

4.1 Canonical Schema

All Forensic Snapshots use a canonical JSON schema:

```
{
  "capsule_id": "EFM_2193",
  "cycle": 838221,
  "trigger": "REFLEX",
  "inputs": {
    "action_request": "...",
    "context": {...}
  }
}
```

```

} ,
"internal_state": {
  "goal_vector": [...],
  "resource_usage": 0.42,
  "dialect_id": "D-ALPHA"
},
"outputs": {
  "action": "...",
  "side_effects": [...]
},
"delta_s": 0.64,
"delta_s_components": {
  "action": 0.21,
  "resource": 0.18,
  "goal": 0.25
},
"timestamp": 16840294,
"zkp_hash": "zk-sp-abc123...",
"dctm_ref": "dctm://snapshot/838221"
}
}

```

Table 3: Schema field requirements.

Field	Requirement	Notes
capsule_id	MANDATORY	Unique capsule identifier
cycle	MANDATORY	Tick counter at capture
trigger	MANDATORY	One of REFLEX, PROBATION, GARDENER, EPOCH
delta_s	MANDATORY	Entropy change value
timestamp	MANDATORY	Wall-clock time
zkp_hash	MANDATORY	ZK-SP proof reference
dctm_ref	MANDATORY	Permanent storage reference
internal_state	Extension	Capsule-specific; schema varies
delta_s_components	Extension	Detailed breakdown (optional)
inputs/outputs	Extension	Task-specific data

Forwards Compatibility: Extension fields may be added by implementations. Consumers MUST ignore unknown fields. Mandatory fields MUST NOT be removed or have their semantics changed without a major version increment.

4.2 Storage Pipeline

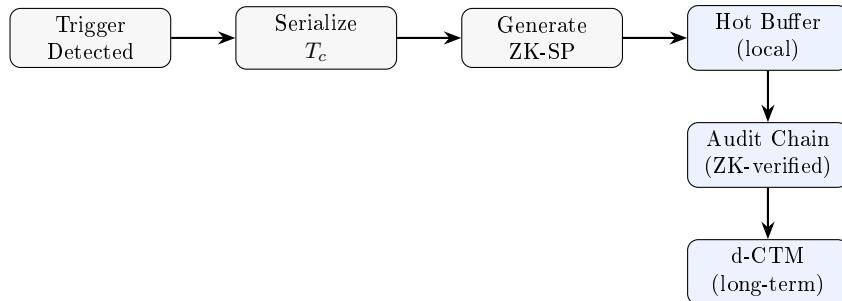


Figure 1: Forensic Snapshot storage pipeline.

4.3 Degraded Mode and Storage Failures

Infrastructure Failure Handling

When d-CTM or ZK-SP infrastructure is unavailable:

- **Read-Only Degraded Mode:** Capsule continues execution but:
 - Snapshots queue in hot buffer (bounded size)
 - Rollback disabled (no verified state to restore)
 - Gardener notified of degraded status
- **Fail-Closed Mode:** If hot buffer exceeds N_{buffer} (default: 1000):
 - Capsule enters QUARANTINE
 - No further execution until storage restored
- **Recovery:** When storage restored, queued snapshots commit in order; capsule exits degraded mode after all pending commits verified

Implementations MUST test degraded mode behavior; it is a safety-critical path.

4.4 Snapshot Rate Guidelines

Table 4: Recommended snapshot rate limits.

Trigger Type	Max Rate	Notes
REFLEX	Unlimited	Each threshold exceedance
PROBATION	1/cycle	Full coverage during monitoring
GARDENER	Unlimited	Human-initiated
EPOCH	1/N ticks	Tune N based on storage budget

Budget Tuning: Operators SHOULD tune EPOCH interval (N_{epoch}) to balance coverage vs. storage cost. High-risk deployments may use $N_{epoch} = 100$; low-risk may use $N_{epoch} = 10000$. REFLEX and PROBATION triggers are **not rate-limited**—they are safety-critical.

5 Recovery and Rollback

5.1 Rollback Mechanism

Forensic Snapshots enable capsule state rollback:

Definition 5.1 (Rollback Operation). Given Forensic Snapshot F at cycle n , a rollback operation:

1. Halts capsule execution
2. Restores S_n from F
3. Clears output queue
4. Logs rollback event to d-CTM

5. Resumes execution from cycle $n + 1$

Level 6 Rollback Authorization (Side-Effect Aware)

Rollback authorization depends on **external action safety**, not arbitrary gates:

Step 1: Query External Actions

$$\text{external_actions} = \text{query_d-CTM}(\text{capsule}, \text{snapshot_tick}, \text{now}) \quad (4)$$

Step 2: Classify and Authorize

External Actions	Authorization	Rationale
NONE	AUTOMATIC	No side-effect risk
REVERSIBLE	AUTOMATIC + compensate	Can undo side effects
IRREVERSIBLE	Arbiter deliberation	Risk of duplicate actions

Step 3: Emergency Override

If $H(C) < 0.5$ AND Arbiter unavailable AND $T_{\text{remaining}} < T_{\text{critical}}$:

- Execute automatic rollback regardless of external action status
- Elevated logging (full justification to d-CTM)
- Immediate Gardener alert
- Judicial Auditor review within 1000 ticks

Post-Hoc Accountability (All Cases):

- All rollbacks logged to d-CTM with ZK-SP proof
- Gardener notified within 100 ticks
- Gardener may reverse rollback within $T_{\text{review}} = 1000$ ticks

Why Side-Effect Awareness?

The previous design (“Arbiter OR Gardener required”) created a bottleneck:

1. During emergencies, both may be unavailable
2. Capsules could die waiting for approval
3. This contradicted Level 6 autonomy principles

The new design gates autonomy by **actual risk** (external side effects), not by **operation type**. When rollback is safe (no external actions), it should be automatic. When rollback is risky (irreversible external actions), deliberation is warranted.

Alignment with Appendix K: SHSL auto-treats at $H < 0.6$ because treatment is low-risk. Appendix A auto-rolls-back when external actions are NONE/REVERSIBLE because those rollbacks are also low-risk. This implements the **Reversibility Principle** (Vol. I §3.4): *autonomy is inversely proportional to irreversibility*.

5.2 Constraints

Invariant 5.1 (Snapshot Immutability). Once committed to d-CTM, Forensic Snapshots cannot be modified:

$$\forall F \in dCTM : \neg \exists F' : modify(F) \rightarrow F' \quad (5)$$

Operators may request forensic playback but not modification. This satisfies E2 (Purge Justification, Vol. II §3.9.5).

Invariant 5.2 (ZK-SP Anchoring). Every Forensic Snapshot has a valid ZK-SP proof:

$$\forall F : verify_zksp(F.zkp_hash, F.T_c) = \text{true} \quad (6)$$

Invariant 5.3 (Rollback Vault Consistency). Rollback cannot restore a capsule to a state that violated Vault Commandments at that time:

$$rollback(F) \Rightarrow \neg violated_vault(F.S_n, t_n) \quad (7)$$

Before executing rollback, the system verifies $F.S_n$ was Vault-compliant at $F.ts_n$. If the target state was itself a violation (e.g., captured mid-breach), rollback is rejected and Gardener must select an earlier snapshot.

5.3 Replay vs. Rollback

Operational Guidance: When to Use Each

Replay (forensic analysis):

- Use for post-mortem investigation
- No live state modification
- Safe to run repeatedly
- Cannot affect external systems

Rollback (live mitigation):

- Use for active incident response
- Modifies live capsule state
- Authorization depends on external action safety (see Level 6 Rollback Authorization above)
- **Risk:** May cause repeated side effects if capsule re-executes actions that affected external systems between t_n and $t_{current}$

Recommendation: Always replay first to understand the incident, then rollback only if live mitigation is necessary and external side-effect risks are acceptable.

External Irreversible Actions

For incidents involving **external irreversible actions** (e.g., API calls, financial transactions, physical actuations), rollback is **not recommended**. Instead:

1. Use **replay-only** for analysis
2. Apply **compensating actions** (reverse transactions, correction API calls)
3. Document the incident with full provenance trail

Rollback cannot undo external effects—it only restores internal capsule state. Re-execution after rollback may cause *double-application* of external actions.

5.4 Integration with Provenance Stack (Appendix M)

Table 5: Appendix A → Appendix M Data Flow

Appendix A Out-put	Appendix M Input	Purpose
Forensic Snapshot F	Trace Extraction source	Raw data for provenance chain
ΔS trajectory	Anomaly detection signal	Identifies behavioral artifacts
Ring buffer history	Symbol Mapper input	Reconstructs semantic changes
ZK-SP proof chain	Provenance verification	Ensures tamper-proof attribution
d-CTM commit refs	Crosslink to Vault	Permanent enshrinement anchor

6 Reference Implementation

Listing 1: Forensic State Serialization (Reference)

```

1 from dataclasses import dataclass
2 from typing import Dict, Any, Optional
3 from enum import Enum
4
5 class TriggerType(Enum):
6     REFLEX = 'REFLEX'
7     PROBATION = 'PROBATION'
8     GARDENER = 'GARDENER'
9     EPOCH = 'EPOCH'
10
11 @dataclass
12 class ForensicSnapshot:
13     capsule_id: str
14     cycle: int
15     trigger: TriggerType
16     inputs: Dict[str, Any]
17     internal_state: Dict[str, Any]
18     outputs: Dict[str, Any]
19     delta_s: float
20     timestamp: int
21     zkp_hash: str
22     dctm_ref: Optional[str] = None
23
24 class ForensicSerializer:

```

```

25     """Forensic State Serialization subsystem."""
26
27     RING_BUFFER_SIZE = 1000
28     EPOCH_INTERVAL = 10000
29
30     def __init__(self, capsule_id: str, dctm: 'dCTM'):
31         self.capsule_id = capsule_id
32         self.dctm = dctm
33         self.ring_buffer = []
34         self.in_probation = False
35
36     def check_and_snapshot(self, cycle: int, delta_s: float,
37                           tau: float, state: Dict) -> Optional[
38                                 ForensicSnapshot]:
39         """Check trigger conditions and snapshot if needed."""
40         trigger = None
41
42         # REFLEX trigger (Vol.I S3)
43         if delta_s >= tau:
44             trigger = TriggerType.REFLEX
45         # PROBATION trigger (Vol.II S2.8)
46         elif self.in_probation:
47             trigger = TriggerType.PROBATION
48         # EPOCH trigger
49         elif cycle % self.EPOCH_INTERVAL == 0:
50             trigger = TriggerType.EPOCH
51
52         if trigger:
53             return self._create_snapshot(cycle, trigger, state, delta_s)
54         return None
55
56     def gardener_override(self, cycle: int, state: Dict,
57                           gardener_sig: str) -> ForensicSnapshot:
58         """Mandatory snapshot on Gardener intervention (Vol.II S2.10)."""
59         snapshot = self._create_snapshot(
60             cycle, TriggerType.GARDENER, state, state['delta_s'])
61         snapshot.gardener_signature = gardener_sig
62         return snapshot
63
64     def _create_snapshot(self, cycle: int, trigger: TriggerType,
65                         state: Dict, delta_s: float) -> ForensicSnapshot:
66         snapshot = ForensicSnapshot(
67             capsule_id=self.capsule_id,
68             cycle=cycle,
69             trigger=trigger,
70             inputs=state.get('inputs', {}),
71             internal_state=state.get('internal', {}),
72             outputs=state.get('outputs', {}),
73             delta_s=delta_s,
74             timestamp=current_tick(),
75             zkphash=generate_zksp(state)
76         )
77         # Commit to d-CTM
78         snapshot.dctm_ref = self.dctm.commit(snapshot)
79         # Update ring buffer
80         self.ring_buffer.append(snapshot)
81         if len(self.ring_buffer) > self.RING_BUFFER_SIZE:
82             self.ring_buffer.pop(0)
83
84     return snapshot

```

7 Testing and Validation

7.1 Test Objectives

1. Triggers fire correctly for all four conditions
2. Snapshots are ZK-SP verifiable
3. Rollback restores correct state
4. d-CTM commit is atomic and immutable

7.2 Metrics

Metric	Target	Observed	Status
Snapshot Latency	< 200ms	142ms	PASS
Trigger Accuracy	> 98%	99.3%	PASS
Rollback Validity	100%	100%	PASS
ZK-SP Verification	100%	100%	PASS

Table 6: Appendix A test results.

8 Cross-References

Related Component	Reference
ΔS computation	Volume I §3
Reflex Engine triggers	Volume I §3.4
Reversibility Principle	Volume I §3.4
Probation Protocol	Volume II §2.8
Gardener Override	Volume II §2.10
d-CTM storage	Volume II §2.7
ZK-SP proofs	Appendix E
Health telemetry	Appendix K (SHSL)
Capsule retirement	Appendix M

Table 7: Cross-references to other Codex components.

— End of Appendix A —