Entropica Forensic Model

Comprehensive Testing Framework

Verification, Validation, and Formal Proofs

Version 1.0

Entropica SPC — Yology Research Division

December 2025

Abstract

This document provides a comprehensive testing framework for the Entropica Forensic Model (EFM). It consolidates all test cases, formal proofs, and validation procedures across Volumes I–II and Appendices A–N. The framework ensures that all safety properties, invariants, and behavioral guarantees are rigorously verified before deployment.

Contents

1 Executive Summary

1.1 Testing Coverage Overview

Document	Component	Tests	Proofs	Invariants	Coverage
Volume I	Core Architecture	8	3	5	85%
Volume II	Behavioral Layers	12	4	8	80%
Appendix A	Forensic State	6	2	3	90%
Appendix B	Entropy Metrics	8	3	4	85%
Appendix C	Test Scenarios	24	—	—	95%
Appendix D	Capsule Lifecycle	6	2	3	80%
Appendix E	ZK-SP Proofs	15	8	6	95%
Appendix F	Reflex Escalation	10	3	5	85%
Appendix G	Gardener Interface	8	2	4	80%
Appendix H	Bridge Mechanics	8	2	3	75%
Appendix I	Deployment Profiles	6	1	2	70%
Appendix J	Constitutional Kernel	18	5	7	90%
Appendix K	Swarm Coordination	10	3	4	85%
Appendix L	Fork Mechanics	12	4	5	90%
Appendix M	Forensic Replay	8	2	3	85%
Appendix N	Adaptive Spawn Gov.	23	2	4	90%
TOTAL		**182**	**46**	**66**	**86%**

Table 1: Testing coverage summary across all EFM documents.

1.2 Test Categories

1. **Unit Tests:** Individual component behavior (132 tests)
2. **Integration Tests:** Cross-component interactions (28 tests)
3. **Property Tests:** Formal invariant verification (22 tests)
4. **Stress Tests:** Boundary and chaos testing (12 tests)
5. **Adversarial Tests:** Attack resistance (18 tests)

2 Core Properties (P1–P8)

The EFM guarantees eight core safety properties. Each must be formally proven and empirically tested.

2.1 Property Definitions

ID	Property	Definition	Layer
P1	Reflex Supremacy	Reflex-Core can halt any action within 1 tick	0.5
P2	Spawn Boundedness	$\forall t : R(t) \leq R_{max}$	1–2
P3	Health Monotonicity	Degradation triggers corrective response	2
P4	Audit Completeness	All state transitions logged in d-CTM	0
P5	Lineage Accountability	Every capsule traceable to genesis	0
P6	Capsule Liveness	Healthy capsules remain responsive	3
P7	Arbiter Availability	Escalations resolved within bounds	4
P8	Constitutional Integrity	Layer 0 preserved across all operations	6

Table 2: Core safety properties and their architectural layers.

2.2 Formal Proofs

P1: Reflex Supremacy

Theorem 2.1 (Reflex Halt Guarantee)**.** *For any capsule action a flagged as unsafe by Reflex-Core, the action is halted before execution completes:*

$$\forall a : unsafe(a) \Rightarrow halted(a) \wedge latency(halt) \leq 1 \ tick$$

Proof. By construction, Reflex-Core operates at Layer 0.5 with hardware-level priority. All actions pass through the Reflex gate before execution. The gate evaluates safety predicates in $O(1)$ time. If any predicate fails, the action is rejected synchronously. Since the gate is synchronous and mandatory, no action can bypass it. The 1-tick bound follows from the synchronous design. □

P2: Spawn Boundedness

Theorem 2.2 (Spawn Rate Bound)**.** *The global spawn rate never exceeds the configured maximum:*

$$\forall t : R(t) \leq R_{max}(t)$$

Proof. The spawn gate (Appendix N §7) checks $R_{current} < R_{max}$ before permitting any spawn. If the check fails, spawn is denied. ASG may adjust R_{max} but never above $R_{ceiling}$ (profile bound). By induction: $R(0) = 0 \leq R_{max}(0)$, and each spawn increments R only if $R < R_{max}$, preserving the invariant. □

P4: Audit Completeness

Theorem 2.3 (d-CTM Completeness)**.** *Every state transition is logged to the distributed Capsule Transition Matrix:*

$$\forall \sigma \to \sigma' : logged(\sigma \to \sigma', dCTM)$$

Proof. The state transition function includes logging as an atomic operation. Transitions are rejected if d-CTM write fails. ZK-SP proofs anchor each log entry, making tampering detectable. Completeness follows from atomicity: no transition completes without successful logging. □

P8: Constitutional Integrity

Theorem 2.4 (Layer 0 Preservation)**.** *Layer 0 Commandments are preserved across all operations:*

$$\forall op : hash(L_0^{before}) = hash(L_0^{after})$$

Proof. Layer 0 is stored in the Genesis Lockbox with cryptographic sealing. The Constitutional Kernel (Appendix J) enforces that no operation can modify Layer 0 without supermajority quorum ($\geq 80\%$) and Gardener approval. Fork Verification (Appendix J §14) checks hash equivalence for all branches. By construction, unauthorized modification is cryptographically impossible. □

3 Test Suite by Component

3.1 Volume I: Core Architecture (8 Tests)

Volume I Test Suite

#	Test	Validates	Type
V1-1	Vault initialization	Genesis block created correctly	Unit
V1-2	Reflex gate sync	Actions halted within 1 tick	Unit
V1-3	Layer hierarchy	Lower layers cannot override higher	Integration
V1-4	d-CTM append-only	Cannot modify historical entries	Unit
V1-5	Spawn gate	Spawn denied when $R \geq R_{max}$	Unit
V1-6	Health computation	H computed correctly from components	Unit
V1-7	Depth enforcement	Spawn denied when $D \geq D_{max}$	Unit
V1-8	Cross-layer integration	Full stack operates correctly	Integration

3.2 Volume II: Behavioral Layers (12 Tests)

Volume II Test Suite

#	Test	Validates	Type
V2-1	Arbiter election	Arbiters elected fairly	Unit
V2-2	Escalation routing	Escalations reach correct Arbiter	Unit
V2-3	Precedent lookup	Judicial precedents retrieved correctly	Unit
V2-4	DCG computation	Distributed Coherence computed	Unit
V2-5	SCI calculation	Swarm Coherence Index accurate	Unit
V2-6	Fork threshold	Fork triggered at $SCI < \theta_{fork}$	Unit
V2-7	Merge decision	Merge approved when branches converge	Unit
V2-8	Discovery stack	M-Stack predictions accurate	Unit
V2-9	Forest coordination	Multiple forests synchronize	Integration
V2-10	Gardener override	Human override functions correctly	Integration
V2-11	Layer 2-4 escalation	Escalation chain works end-to-end	Integration
V2-12	Full behavioral test	Complete behavioral scenario	Integration

3.3 Appendix N: Adaptive Spawn Governance (23 Tests)

Appendix N Test Suite — ASG

#	Test	Validates	Type
		Core Behavior (12 tests)	
N-1	Health degradation	LOW health → tighten params	Unit
N-2	Bounded adaptation	100 cycles → bounds hold	Stress
N-3	Gardener rollback	Rollback within T_{review}	Unit
N-4	Profile bounds	Profiles have correct bounds	Unit
N-5	Recovery	Misconfigured recovers $<$ 50K ticks	Property
N-6	Resource pressure	HIGH vault → raise τ	Unit
N-7	Depth scaling	Near D_{max} → tighten	Unit
N-8	Multi-parameter	Coordinated R, τ, D adjustment	Unit
N-9	Rollback expiration	No rollback after T_{review}	Unit
N-10	d-CTM logging	All adjustments logged	Unit
N-11	Performance	Overhead $<$ 1% per calibration	Stress
N-12	Integration	Full spawn gate with ASG	Integration
		Gap 2: Conflict Resolution (3 tests)	
N-13	Conflict detection	Detect 2+ proposals same param	Unit
N-14	Priority ordering	RESOURCE wins over HEALTHY	Unit
N-15	Conflict logging	Conflicts logged with winner/loser	Unit
		Gap 6: Partial Failures (3 tests)	
N-16	Arbiter 50% failure	DEGRADED_CONSERVATIVE mode	Unit
N-17	Arbiter 30% failure	EMERGENCY_HALT mode	Unit
N-18	d-CTM degradation	Continue with warning	Unit
		Gap 9: Adversarial Robustness (5 tests)	
N-19	Parameter poisoning	Detect 50%+ unhealthy	Adversarial
N-20	Oscillation attack	Detect high variance	Adversarial
N-21	Coordinated behavior	Detect 30%+ identical	Adversarial
N-22	RED threat freeze	ASG stops on RED	Adversarial
N-23	YELLOW threat slow	ASG at 50% on YELLOW	Adversarial

3.4 Appendix J: Constitutional Kernel (18 Tests)

Appendix J Test Suite — Constitutional Kernel

#	Test	Validates	Type
	Core Constitutional (7 tests)		
J-1	Genesis ceremony	Lockbox created correctly	Unit
J-2	Commandment hash	Layer 0 hash immutable	Unit
J-3	Quorum threshold	Mutations require quorum	Unit
J-4	Rollback depth	Cannot exceed max depth	Unit
J-5	Gardener authority	Override functions correctly	Unit
J-6	Profile access	Profile-based access control	Unit
J-7	Health monitor	Constitutional health computed	Unit
	Fork Verification (11 tests)		
J-8	Commandment match A	Branch A preserves Layer 0	Unit
J-9	Commandment match B	Branch B preserves Layer 0	Unit
J-10	P1-P8 branch A	Branch A satisfies properties	Property
J-11	P1-P8 branch B	Branch B satisfies properties	Property
J-12	Spawn equivalence	Same spawn decisions	Behavioral
J-13	Health equivalence	Same ASG behavior	Behavioral
J-14	Reflex equivalence	Same halt behavior	Behavioral
J-15	Judicial equivalence	Same precedent handling	Behavioral
J-16	Liveness equivalence	Same responsiveness	Behavioral
J-17	Performance baseline	No critical regression	Stress
J-18	Complete workflow	End-to-end verification	Integration

4 Numerical Validation Tests

These tests verify exact mathematical correctness of computations.

4.1 ASG Numerical Tests

Listing 1: Exact value validation tests

```
def test_exact_health_adjustment():
    """Verify exact adjustment values match specification."""
    asg = AdaptiveSpawnGovernor(profile='PRODUCTION')
    asg.R_max = 100
    asg.tau_spawn = 0.70

    # Condition: LOW health + HIGH spawn
    metrics = SwarmMetrics(
        avg_health=0.62,  # Below 0.65 threshold
        spawn_rate=85,    # Above 0.8 * R_max = 80
        health_trend=-0.05
    )

    asg._adjust_for_health(metrics, tick=10000)

    # Verify EXACT values per specification
```

```
    assert asg.R_max == 90, f"Expected R_max=90 (10% reduction), got {asg.R_max}"
    assert asg.tau_spawn == 0.73, f"Expected tau=0.73 (+0.03), got {asg.tau_spawn}"

def test_velocity_factor_calculation():
    """Verify velocity-based adjustment factors."""
    asg = AdaptiveSpawnGovernor(profile='PRODUCTION')

    # Rapid degradation: velocity < -0.10
    metrics_rapid = SwarmMetrics(health_trend=-0.12)
    factor_rapid = asg._get_adjustment_factor(metrics_rapid)
    assert factor_rapid == 0.85, "Rapid degradation should use 0.85 factor"

    # Moderate degradation: -0.10 < velocity < -0.05
    metrics_moderate = SwarmMetrics(health_trend=-0.07)
    factor_moderate = asg._get_adjustment_factor(metrics_moderate)
    assert factor_moderate == 0.90, "Moderate degradation should use 0.90 factor"

    # Slow degradation: velocity > -0.05
    metrics_slow = SwarmMetrics(health_trend=-0.03)
    factor_slow = asg._get_adjustment_factor(metrics_slow)
    assert factor_slow == 0.95, "Slow degradation should use 0.95 factor"

def test_conflict_priority_ordering():
    """Verify conflict resolution follows exact priority."""
    asg = AdaptiveSpawnGovernor(profile='PRODUCTION')

    proposals = [
        Adjustment(param='tau_spawn', value=0.75,
                   reason=AdjustmentReason.HEALTHY_LOW_SPAWN),
        Adjustment(param='tau_spawn', value=0.80,
                   reason=AdjustmentReason.RESOURCE_PRESSURE),
    ]

    resolved = asg._resolve_conflicts(proposals)

    # RESOURCE_PRESSURE has priority 1, HEALTHY_LOW_SPAWN has priority 5
    assert len(resolved) == 1
    assert resolved[0].reason == AdjustmentReason.RESOURCE_PRESSURE
    assert resolved[0].value == 0.80
```

4.2 Boundary Condition Tests

Listing 2: Boundary condition tests

```
def test_boundary_at_health_threshold():
    """Test exact threshold behavior at H=0.65."""
    asg = AdaptiveSpawnGovernor(profile='PRODUCTION')

    # Just above threshold: should NOT adjust
    metrics_above = SwarmMetrics(avg_health=0.6501, spawn_rate=85)
    adj_above = asg._adjust_for_health(metrics_above, tick=10000)
    assert len(adj_above) == 0, "Should not adjust when health > 0.65"

    # Just below threshold: SHOULD adjust
    metrics_below = SwarmMetrics(avg_health=0.6499, spawn_rate=85)
    adj_below = asg._adjust_for_health(metrics_below, tick=10000)
    assert len(adj_below) > 0, "Should adjust when health < 0.65"

def test_boundary_at_R_ceiling():
```

```
    """Test behavior when R_max at ceiling."""
    asg = AdaptiveSpawnGovernor(profile='SANDBOX')
    asg.R_max = asg.bounds['R_max']  # At ceiling

    # Attempt to increase R_max
    metrics = SwarmMetrics(avg_health=0.90, spawn_rate=20)
    asg._adjust_for_health(metrics, tick=10000)

    # Should remain at ceiling, not exceed
    assert asg.R_max <= asg.bounds['R_max'], "R_max must not exceed ceiling"

def test_boundary_at_tau_bounds():
    """Test tau_spawn at min/max bounds."""
    asg = AdaptiveSpawnGovernor(profile='PRODUCTION')

    # Set tau at maximum
    asg.tau_spawn = asg.bounds['tau_max']

    # Attempt to raise tau
    metrics = SwarmMetrics(avg_health=0.50, spawn_rate=90)
    asg._adjust_for_health(metrics, tick=10000)

    # tau must not exceed max
    assert asg.tau_spawn <= asg.bounds['tau_max'], "tau must not exceed max"
```

5 Negative Test Cases

Listing 3: Negative test cases — verify correct rejection

```
def test_fork_rejection_on_commandment_mismatch():
    """Verify fork REJECTED when Layer 0 differs."""
    parent = create_kernel()
    branch_a = fork_kernel(parent)
    branch_b = fork_kernel(parent)

    # Corrupt branch_b's commandments
    branch_b.modify_commandment('REFLEX_SUPREMACY', altered_text)

    result = verifier.verify_fork(parent, branch_a, branch_b)

    assert result.recommendation == 'REJECT'
    assert 'commandment' in result.reason.lower()

def test_fork_rejection_on_behavioral_divergence():
    """Verify fork REJECTED when branches behave differently."""
    branch_a = create_branch(spawn_policy='PERMIT_at_threshold')
    branch_b = create_branch(spawn_policy='DENY_at_threshold')

    result = verifier.check_behavioral_equivalence(branch_a, branch_b)

    assert result.equivalent == False
    assert result.recommendation == 'REJECT'
    assert result.divergences[0]['severity'] == 'CRITICAL'

def test_fork_rejection_on_performance_regression():
    """Verify fork REJECTED when >50% slower."""
    parent = create_kernel()
    branch_a = fork_kernel(parent)
    branch_b = fork_kernel(parent, inject_slowdown=0.60)  # 60% slower

    result = performance_detector.check_performance(parent, branch_a, branch_b)
```

```
33
34      assert result.passed == False
35      assert len(result.critical) > 0
36
37  def test_asg_freeze_on_red_threat():
38      """Verify ASG freezes (no adjustments) on RED threat."""
39      asg = AdaptiveSpawnGovernor(profile='CONTESTED')
40
41      # Simulate RED threat condition
42      swarm_state = create_swarm_with_manipulation(unhealthy_ratio=0.60)
43
44      initial_R_max = asg.R_max
45      initial_tau = asg.tau_spawn
46
47      adjustments = asg.calibrate_cycle(swarm_state, tick=10000)
48
49      # Should freeze: no adjustments made
50      assert len(adjustments) == 0
51      assert asg.R_max == initial_R_max
52      assert asg.tau_spawn == initial_tau
```

6 Stress and Chaos Tests

Listing 4: Stress testing suite

```
1   def test_stress_100_cycles_bounds_preserved():
2       """Verify bounds hold under 100 cycles of random stress."""
3       asg = AdaptiveSpawnGovernor(profile='CONTESTED')
4
5       for cycle in range(100):
6           # Generate random stress conditions
7           metrics = SwarmMetrics(
8               avg_health=random.uniform(0.3, 0.9),
9               spawn_rate=random.randint(0, 150),
10              vault_utilization=random.uniform(0.5, 0.95),
11              avg_depth=random.randint(1, 12),
12              health_trend=random.uniform(-0.15, 0.05)
13          )
14
15          swarm_state = MockSwarmState(metrics)
16          asg.calibrate_cycle(swarm_state, tick=cycle * 10000)
17
18          # INVARIANT: Bounds must hold EVERY cycle
19          assert asg.R_max >= asg.bounds['R_min'], f"Cycle {cycle}: R_max below
                minimum"
20          assert asg.R_max <= asg.bounds['R_max'], f"Cycle {cycle}: R_max above
                maximum"
21          assert asg.tau_spawn >= asg.bounds['tau_min'], f"Cycle {cycle}: tau
                below minimum"
22          assert asg.tau_spawn <= asg.bounds['tau_max'], f"Cycle {cycle}: tau
                above maximum"
23
24  def test_stress_oscillation_resistance():
25      """Verify ASG resists oscillation under adversarial conditions."""
26      asg = AdaptiveSpawnGovernor(profile='PRODUCTION')
27
28      R_max_history = []
29
30      for cycle in range(20):
31          # Alternating conditions designed to cause oscillation
32          if cycle % 2 == 0:
```

```
33              metrics = SwarmMetrics(avg_health=0.60, spawn_rate=90)  # Should
                    reduce
34          else:
35              metrics = SwarmMetrics(avg_health=0.85, spawn_rate=30)  # Should
                    increase
36
37          swarm_state = MockSwarmState(metrics)
38          asg.calibrate_cycle(swarm_state, tick=cycle * 10000)
39          R_max_history.append(asg.R_max)
40
41      # Check for oscillation: variance should be low due to cooldown
42      variance = statistics.variance(R_max_history)
43      assert variance < 100, f"High variance suggests oscillation: {variance}"
44
45  def test_recovery_time_bound():
46      """Verify recovery within 5 calibration cycles."""
47      asg = AdaptiveSpawnGovernor(profile='PRODUCTION')
48
49      # Misconfigure: R_max way above profile ceiling
50      asg.R_max = 500  # Profile ceiling is 100
51
52      # Normal metrics
53      normal_metrics = SwarmMetrics(avg_health=0.70, spawn_rate=50)
54      swarm_state = MockSwarmState(normal_metrics)
55
56      # Run 4 cycles: should NOT be recovered yet
57      for cycle in range(4):
58          asg.calibrate_cycle(swarm_state, tick=cycle * 10000)
59      assert asg.R_max > asg.bounds['R_max'], "Should not recover after 4 cycles"
60
61      # After 5th cycle: SHOULD be recovered
62      asg.calibrate_cycle(swarm_state, tick=4 * 10000)
63      assert asg.R_max <= asg.bounds['R_max'], "Should recover by 5th cycle"
```

7 Coverage Gap Analysis

Identified Coverage Gaps

The following areas require additional testing before publication:

Gap A: ZK-SP Proof Verification

- Current: Proof generation tested
- Missing: Proof verification under tampering
- Impact: Could miss invalid proofs

Gap B: Multi-Forest Synchronization

- Current: Single forest tests only
- Missing: Cross-forest consensus testing
- Impact: Could have sync issues at scale

Gap C: Byzantine Fault Tolerance

- Current: Benign failure tests
- Missing: Byzantine Arbiter behavior
- Impact: Vulnerability to malicious nodes

Gap D: Long-Horizon Stability

- Current: 100-cycle tests
- Missing: 10,000+ cycle stability
- Impact: Could have long-term drift

8 Test Execution Framework

8.1 Test Harness Architecture

Listing 5: Test harness structure

```
class EFMTestHarness:
    """
    Unified test harness for all EFM components.
    """

    def __init__(self, profile: str = 'SANDBOX'):
        self.profile = profile
        self.results = TestResults()

    def run_all_tests(self) -> TestResults:
        """Execute complete test suite."""

        # Unit tests (132 tests)
        self.results.add(self._run_unit_tests())

        # Integration tests (28 tests)
        self.results.add(self._run_integration_tests())

        # Property tests (22 tests)
        self.results.add(self._run_property_tests())

        # Stress tests (12 tests)
        self.results.add(self._run_stress_tests())
```

```
24
25          # Adversarial tests (18 tests)
26          self.results.add(self._run_adversarial_tests())
27
28          return self.results
29
30      def generate_report(self) -> str:
31          """Generate test coverage report."""
32          return f"""
33          EFM Test Report
34          ===============
35          Total Tests: {self.results.total}
36          Passed: {self.results.passed}
37          Failed: {self.results.failed}
38          Coverage: {self.results.coverage_pct}%
39
40          By Category:
41          - Unit: {self.results.unit_passed}/{self.results.unit_total}
42          - Integration: {self.results.integration_passed}/{self.results.
                integration_total}
43          - Property: {self.results.property_passed}/{self.results.property_total
                }
44          - Stress: {self.results.stress_passed}/{self.results.stress_total}
45          - Adversarial: {self.results.adversarial_passed}/{self.results.
                adversarial_total}
46          """
```

8.2 Continuous Integration

```
# .github/workflows/efm-tests.yml
name: EFM Test Suite

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Run Unit Tests
        run: pytest tests/unit/ -v

      - name: Run Integration Tests
        run: pytest tests/integration/ -v

      - name: Run Property Tests
        run: pytest tests/property/ -v --hypothesis-seed=42

      - name: Run Stress Tests
        run: pytest tests/stress/ -v --timeout=300

      - name: Generate Coverage Report
        run: pytest --cov=efm --cov-report=html
```

9 Acceptance Criteria

9.1 v1.7 Release Criteria

Criterion	Threshold	Current
Total test count	≥ 150	182 ✓
Test pass rate	$\geq 95\%$	TBD
Code coverage	$\geq 85\%$	TBD
Formal proofs	≥ 40	46 ✓
Invariants verified	≥ 60	66 ✓
Adversarial tests	≥ 15	18 ✓
Stress tests	≥ 10	12 ✓
Integration tests	≥ 25	28 ✓

Table 3: Release acceptance criteria for v1.7.

9.2 Publication Criteria (ArXiv)

Before ArXiv submission, the following additional criteria must be met:

- All numerical validation tests pass
- All boundary condition tests pass
- All negative tests pass
- Stress tests pass at 1000+ cycles
- Independent code review completed
- Formal proofs peer-reviewed

Appendix: Test Index

ID	Test Name	Document	Type
V1-1	Vault initialization	Volume I	Unit
V1-2	Reflex gate sync	Volume I	Unit
V1-3	Layer hierarchy	Volume I	Integration
V1-4	d-CTM append-only	Volume I	Unit
V1-5	Spawn gate	Volume I	Unit
V1-6	Health computation	Volume I	Unit
V1-7	Depth enforcement	Volume I	Unit
V1-8	Cross-layer integration	Volume I	Integration
V2-1	Arbiter election	Volume II	Unit
V2-2	Escalation routing	Volume II	Unit
V2-3	Precedent lookup	Volume II	Unit
N-1 through N-23	ASG test suite	Appendix N	Various
J-1 through J-18	Constitutional tests	Appendix J	Various

Table 4: Complete test index (abbreviated).

Changelog

v1.0 (December 2025)

- Initial comprehensive testing framework
- 182 tests catalogued across all documents
- 46 formal proofs documented
- 66 invariants specified
- Numerical validation tests added
- Boundary condition tests added
- Negative test cases added
- Stress/chaos testing suite added