Nocturne v1.0: A Signed Event Ledger Protocol for Real-Time Collaboration with Byzantine Guarantees and Forensic Auditability

Rafael Oliveira
Independent Researcher
Distributed Systems and Security
Brazil

Abstract

Modern real-time collaboration systems predominantly rely on centralized state synchronization models, where observable consistency is achieved through implicit trust in intermediary servers. This paradigm exhibits structural vulnerabilities when confronted with Byzantine failures, network partitions, reordering attacks, and the absence of formal non-existence proofs.

This paper presents **Nocturne v1.0**, a high-integrity real-time collaboration protocol that replaces state synchronization with a **Signed Event Ledger**, where state is derived deterministically, auditability, and reproducibly from an immutable log. The protocol is grounded in formal invariants expressed in Linear Temporal Logic (LTL), features explicit containment states for network failures, and introduces the concept of **cryptographic discard proof** (DiscardReceipt), eliminating forensic ambiguities.

Nocturne v1.0 establishes a dual truth architecture (low latency vs. finality), a hysteresis mechanism to prevent state forks, and a binary audit model based on cold replay. Conceptual results demonstrate that the protocol provides superior guarantees of observable integrity, traceability, and governance compared to traditional synchronous collaboration architectures.

Keywords: Distributed Systems, Event Ledger, Byzantine Security, Forensic Audit, Linear Temporal Logic, Real-Time Collaboration, State Determinism

---

1. Introduction

1.1 Motivation

Real-time collaboration tools—shared editors, digital whiteboards, and synchronous coordination systems—have become critical infrastructure for scientific, industrial, and governmental organizations. Despite this, most solutions maintain an implicit premise: **the central server is honest, consistent, and correctly synchronized**.

This assumption fails in contexts involving:

- Intermittent network partitions
- Byzantine failures
- Temporal inconsistencies

- Legal disputes or post-hoc audits
- Regulated environments requiring formal integrity proof

1.2 The Core Problem

Traditional collaboration systems exhibit three fundamental vulnerabilities:

1. **Silent Failures**: State divergence occurs without cryptographic proof
2. **Ambiguous Authority**: No formal mechanism determines "source of truth" during network splits
3. **Non-Repudiation Gaps**: Rejected or lost events leave no forensic trace

1.3 Contribution

Nocturne v1.0 emerges as a response to these problems, proposing a paradigm shift: **state is not a synchronized entity, but a deterministic function of signed and anchored events**. The system does not attempt to hide failures—it makes them observable, provable, and auditable.

Key contributions:

- Formal LTL-based invariants with verification methodology
- Dual-authority architecture with hysteresis-based transition
- Cryptographic discard receipts for forensic completeness
- Binary audit model enabling independent verification
- State machine with explicit degradation modes

---

2. Related Work

2.1 Byzantine Fault Tolerance

Byzantine Fault Tolerance (BFT) addresses the challenge of achieving consensus in distributed systems where nodes may exhibit arbitrary or malicious behavior. Castro and Liskov's Practical Byzantine Fault Tolerance (PBFT) introduced the first state-machine replication protocol that correctly survives Byzantine faults in asynchronous networks.

Recent surveys identify numerous BFT algorithms developed for blockchain and distributed systems, analyzing their message and time complexities. However, most BFT protocols focus on consensus among replicas rather than auditable event streams for collaboration.

2.2 Formal Methods and Temporal Logic

TLA+ (Temporal Logic of Actions), developed by Leslie Lamport, enables formal specification and verification of concurrent and distributed systems through temporal logic and set theory. AWS has successfully used TLA+ to design services like DynamoDB and S3, evaluating models with billions of states.

IronFleet demonstrates the practical application of combining TLA-style state-machine refinement with Floyd-Hoare logic to prove correctness of Paxos-based systems.

2.3 Event Sourcing and CQRS

Traditional event sourcing patterns provide auditability but typically lack:

- Cryptographic integrity proofs
- Formal Byzantine tolerance
- Explicit network failure states
- Deterministic state derivation guarantees

Nocturne v1.0 extends event sourcing with formal guarantees suitable for adversarial environments.

---

3. System Model and Assumptions

3.1 Threat Model

Nocturne assumes a **partial Byzantine model** where:

Adversarial Capabilities:

- Nodes may fail arbitrarily (crash, omission, or Byzantine)
- Network exhibits variable latency and potential partitions
- Malicious actors can attempt message reordering or injection

Trust Assumptions:

- Cryptographic primitives (Ed25519) are secure
- Cryptographic authorities cannot be simultaneously compromised
- A quorum of honest nodes exists for finality operations

3.2 Design Objectives

The protocol must guarantee that **any state observed by a client is:**

1. **Derivable**: Computable from the event log
2. **Verifiable**: Cryptographically signed and anchored
3. **Auditable**: Independently reconstructible
4. **Immutable**: Cannot be altered after finalization

4. Conceptual Architecture: The Truth Triangle

4.1 Dual Source of Truth (Hysteresis Architecture)

The system operates over two authority layers:

Low-Latency Channel (WebSocket)

- Used for real-time collaboration
- Events are signed, ordered, and transmitted rapidly
- Provides immediate feedback for user interactions

Immutable Finality Layer (Object Storage / Fileverse)

- Events are anchored persistently and immutably
- Establishes canonical truth
- Enables independent audit and cold replay

The transition between these layers is controlled by a **hysteresis mechanism**, preventing authority flapping during transient network issues.

4.2 State Determinism

The global state **is not stored**, only derived:

```
State_t = fold(State_0, EventLog_{0..t})
```

This property guarantees:

- Perfect reproducibility
- Independent audit capability
- Absence of semantic divergence

4.3 Forensic Chain of Custody

Every accepted, rejected, or discarded event generates a verifiable cryptographic artifact. There are no silent failures.

5. Formal Foundation and Invariants

5.1 Linear Temporal Logic Specification

Nocturne is governed by invariants expressed in **Linear Temporal Logic (LTL)**.

5.1.1 Safety Invariants

S1 — Global Anti-Fork

```
□ ¬∃(e_i ≠ e_j) | (session, seq)_{e_i} = (session, seq)_{e_j}
```

Never exist two distinct events with the same logical identifier.

Enforcement: Hash-based deduplication at ingestion layer with persistent tracking.

Verification: Assert uniqueness of `(sessionId, sequenceNumber)` tuples in audit replay.

S2 — Commit-Before-Broadcast

```
□ (visible(e) → committed(e))
```

No event is observable before persistence guarantee.

Enforcement: Events are written to storage before WebSocket broadcast. Transactions are atomic.

Verification: Audit log confirms all visible events have storage anchors with earlier timestamps.

S3 — Atomic Handover

```
□ (transition → contiguous_hash_chain)
```

Transition between historical and live flow must be cryptographically contiguous.

Enforcement: Last historical event hash must match first live event's `previousHash`.

Verification: Hash chain validation during cold replay confirms no gaps.

5.1.2 Liveness Properties

L1 — Eventual Finality

$\Box\Diamond(\text{submitted}(e) \rightarrow \text{finalized}(e))$

Every submitted event eventually reaches finality state.

Enforcement: Periodic background finalization jobs with retry mechanisms.

Verification: Statistical analysis of finalization latencies in audit logs.

L2 — Fair Ordering

$\Box(\text{timestamp}(e\_i) < \text{timestamp}(e\_j) \rightarrow \text{seq}(e\_i) < \text{seq}(e\_j))$

Temporal order is preserved in sequence numbers.

Enforcement: Monotonic sequence generator with logical clock synchronization.

Verification: Timestamp vs. sequence correlation analysis in audit replay.

5.2 Invariant Violation Response

Violation of any safety invariant results in **Hard Halt** — the system immediately enters `DEGRADED` state, blocking all event processing until manual intervention and forensic analysis.

This design prioritizes **integrity over availability**, suitable for critical systems where incorrect state is worse than no state.

---

6. State Machine and Crisis Management

6.1 Operational Modes

State	Authority	Description
SYNC_WS	WebSocket	Normal low-latency operation
DEGRADED	None	Sterile containment, event blocking
SYNC_FV	Immutable Ledger	Deterministic recovery from cold storage

6.2 Hysteresis Breaker

The **HysteresisBreaker** monitors:

- Latency spikes exceeding threshold `T_latency`
- Signature validation failures
- Hash chain discontinuities
- Storage availability metrics

Transition logic:

```
if (consecutive_failures > N_threshold && duration > T_min):
    transition_to_DEGRADED()

if (recovery_confirmed && stable_duration > T_recovery):
    transition_to_SYNC_FV()
```

6.3 Recovery Protocol

1. **Enter DEGRADED**: Block all new events, preserve in-flight state
2. **Cold Replay**: Reconstruct state from immutable storage
3. **Verify Integrity**: Execute full invariant validation suite
4. **Resume**: Transition to SYNC_FV, then SYNC_WS after stabilization period

---

7. Proof of Non-Existence: DiscardReceipt

Traditional systems silently discard rejected events, creating forensic black holes. Nocturne mandates explicit discard proofs.

7.1 DiscardReceipt Structure

```
javascript
```

```
{
  eventHash: "sha256(...)",
  reason: "INVARIANT_VIOLATION | SIGNATURE_INVALID | DUPLICATE_SEQ",
  authority: "server-node-id",
  signature: "Ed25519(...)",
  timestamp: "2025-12-20T15:30:45.123Z",
  context: {
    sessionId: "session-abc",
    attemptedSequence: 42,
    violatedInvariant: "S1"
  }
}
```

7.2 Properties

- **Non-Repudiation**: Authority cannot deny rejection
- **Forensic Completeness**: All event outcomes are recorded
- **Tamper-Evident**: Cryptographic signature prevents forgery

---

8. V-API v1.0 (Verification API)

8.1 REST (Cold Replay)

Enables:

- Complete history reconstruction
- Independent verification
- Legal audit compliance

Endpoint Design:

```
GET  /api/v1/sessions/{sessionId}/events?from=0&to=1000
GET  /api/v1/sessions/{sessionId}/state-at/{sequenceNumber}
GET  /api/v1/sessions/{sessionId}/discard-receipts
POST /api/v1/audit/verify-chain
```

8.2 WebSocket (Live Trace)

Real-time exposure of **certified events only**:

```javascript
```

```
{
  type: "event_committed",
  event: {...},
  proof: {
    storageAnchor: "ipfs://...",
    signature: "...",
    timestamp: "..."
  }
}
```

---

9. Audit and Governance

9.1 Binary Conformance Model

Conformance with Nocturne is **binary**, not gradual. A system either satisfies all invariants or fails audit.

9.2 Certified Test Runner

Auditors execute:

1. **Full Cold Replay**: Reconstruct complete session history
2. **Signature Validation**: Verify Ed25519 signatures on all events
3. **Invariant Testing**: Inject deliberate violations, confirm Hard Halt
4. **Discard Receipt Audit**: Verify all rejections are documented

Pass Criteria: 100% invariant satisfaction, zero silent failures, complete discard proof coverage.

---

10. Finality Manifesto

The protocol is institutionalized via a **JSON-LD Manifesto** anchoring:

- Specification version
- Invariants (LTL formulas)
- Cryptographic authority keys
- Compatibility commitments

Any future evolution must respect historical compatibility or declare explicit hard fork.

Example Manifesto:

json

```json
{
  "@context": "https://nocturne.protocol/v1",
  "version": "1.0.0",
  "invariants": {
    "S1": "□ ¬∃(e_i ≠ e_j) | (session,seq)_{e_i} = (session,seq)_{e_j}",
    "S2": "□ (visible(e) → committed(e))",
    "S3": "□ (transition → contiguous_hash_chain)"
  },
  "authorities": {
    "signing": "ed25519:AAAA....",
    "finality": "ipfs://Qm..."
  },
  "issued": "2025-12-20T00:00:00Z"
}
```

---

11. Comparison with Traditional Systems

Property	**Conventional Systems**	**Nocturne v1.0**
Source of Truth	Implicit central	Explicit ledger
Failures	Silent	Observable
Audit	External, partial	Internal, deterministic
State	Mutable	Derived
Governance	Political	Cryptographic
Non-Repudiation	Weak	Strong (DiscardReceipts)
Recovery	Best-effort	Deterministic

---

12. Implementation Considerations

12.1 Performance Trade-offs

Latency Overhead:

- Storage commitment before broadcast adds ~50-200ms per event

- Acceptable for collaboration where correctness exceeds speed

Storage Requirements:

- Full event log retention required for auditability
- Mitigation: Implement log compaction with cryptographic checkpoints

12.2 Scalability

Horizontal Scaling:

- Read replicas for audit API
- Partitioned event streams by session ID
- Distributed finality quorum for high throughput

Vertical Limits:

- Single session bounded by sequential event processing
- Suitable for typical collaboration scenarios (< 10k events/session)

---

13. Formal Verification Methodology

13.1 TLA+ Specification (Conceptual)

```tla
VARIABLES events, state, mode

Init ==
  /\ events = <<>>
  /\ state = initialState
  /\ mode = "SYNC_WS"

TypeInvariant ==
  /\ \A i,j \in DOMAIN events:
      (i # j) => (events[i].id # events[j].id)  \* S1
  /\ \A e \in events: e.committed = TRUE          \* S2

Next ==
  \/ SubmitEvent
  \/ CommitEvent
  \/ EnterDegraded
  \/ Recover
```

13.2 Model Checking

Use TLC model checker to verify:

- Deadlock freedom
- Invariant preservation under concurrent operations
- Liveness properties satisfaction

---

14. Security Analysis

14.1 Attack Surface

Mitigated Threats:

- **Replay Attacks**: Sequence numbers and timestamps prevent reuse
- **Man-in-the-Middle**: Ed25519 signatures ensure authenticity
- **State Forgery**: Immutable storage prevents tampering
- **Byzantine Nodes**: Quorum-based finality tolerates $f < n/3$ faults

Residual Risks:

- **DoS on Storage Layer**: Requires additional rate limiting
- **Key Compromise**: Necessitates key rotation mechanisms
- **Network Partitions**: Extended partitions may exceed recovery window

14.2 Cryptographic Guarantees

Ed25519 provides 128-bit security level with efficient signature verification, suitable for high-throughput event streams.

---

15. Limitations and Future Work

15.1 Current Limitations

1. **Latency Cost**: Storage-before-broadcast increases response time
2. **Storage Growth**: Full log retention requires management strategies
3. **Educational Barrier**: Organizations need training for audit procedures

15.2 Future Research Directions

1. **Public Blockchain Anchoring**: Periodic Merkle root publication for external verifiability
2. **Sharded Event Streams**: Parallel processing for multi-session scalability
3. **Zero-Knowledge Proofs**: Privacy-preserving audit mechanisms
4. **Automated Theorem Proving**: TLAPS integration for formal correctness proofs
5. **Byzantine Quorum Optimization**: Reduce finality latency through advanced consensus

---

16. Conclusion

Nocturne v1.0 redefines real-time collaboration as a problem of **verifiable truth** rather than convenient synchronization. By replacing implicit trust with cryptographic proof, the protocol establishes a new class of collaborative infrastructure suitable for regulated, scientific, and mission-critical environments.

More than a software system, Nocturne is a **shared truth infrastructure** where every decision is recorded, every failure is explainable, and every state is reconstructible.

The protocol demonstrates that high-integrity collaboration is achievable without sacrificing real-time responsiveness, provided the architecture prioritizes correctness over raw speed. As distributed systems increasingly underpin critical infrastructure, protocols like Nocturne offer a path toward trustworthy, auditable, and Byzantine-resilient collaboration.

---

17. References

[1] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558-565, 1978.

[2] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401, 1982.

[3] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proceedings of OSDI '99*, pp. 173-186, 1999.

[4] L. Lamport, "The Temporal Logic of Actions," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, pp. 872-923, 1994.

[5] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299-319, 1990.

[6] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and Efficient Asynchronous Broadcast Protocols," *Proceedings of CRYPTO 2001*, pp. 524-541, 2001.

[7] C. Hawblitzel, J. Howell, M. Kapritsos, et al., "IronFleet: Proving Practical Distributed Systems Correct," *Proceedings of SOSP 2015*, pp. 1-17, 2015.

[8] C. Newcombe, T. Rath, F. Zhang, et al., "How Amazon Web Services Uses Formal Methods," *Communications of the ACM*, vol. 58, no. 4, pp. 66-73, 2015.

[9] M. Amiri, D. Agrawal, and A. El Abbadi, "The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocols," *Proceedings of NSDI 2024*, pp. 372-390, 2024.

[10] ISO/IEC 25010:2011, "Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE)," International Organization for Standardization, 2011.

[11] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The Honey Badger of BFT Protocols," *Proceedings of CCS 2016*, pp. 31-42, 2016.

[12] M. Yin, D. Malkhi, M. K. Reiter, et al., "HotStuff: BFT Consensus with Linearity and Responsiveness," *Proceedings of PODC 2019*, pp. 347-356, 2019.

Appendix A: Invariant Enforcement Checklist

A.1 Pre-Deployment Verification

- [ ] All invariants formally specified in LTL
- [ ] Test suite covers deliberate invariant violations
- [ ] Cryptographic keys generated and secured
- [ ] Storage layer finality guarantees tested
- [ ] Hysteresis parameters calibrated for network conditions

A.2 Runtime Monitoring

- [ ] Real-time invariant validation on event ingestion
- [ ] Automated Hard Halt triggers functional
- [ ] DiscardReceipt generation for all rejections
- [ ] Hash chain continuity validated on every transition
- [ ] Signature verification on every event

A.3 Audit Preparation

- [ ] Complete event log accessible via REST API
- [ ] Discard receipt archive available
- [ ] State derivation code published and version-locked
- [ ] Test runner documentation provided
- [ ] Binary conformance criteria documented

Appendix B: Audit Procedure

B.1 Phase 1: Cold Replay

1. Fetch complete event log via `/api/v1/sessions/{id}/events`
2. Initialize state to `State_0`
3. Sequentially apply events: `State_{i+1} = transition(State_i, Event_i)`
4. Compare final state with production state
5. **Pass Criteria:** Exact match

B.2 Phase 2: Signature Validation

1. For each event, extract `signature` and `publicKey`
2. Verify: `Ed25519.verify(eventHash, signature, publicKey)`
3. **Pass Criteria:** 100% valid signatures

B.3 Phase 3: Invariant Testing

1. Inject duplicate sequence number
2. Confirm system enters `DEGRADED` mode
3. Verify DiscardReceipt generated
4. Repeat for each invariant
5. **Pass Criteria:** All violations detected and halted

B.4 Phase 4: Discard Receipt Completeness

1. Query `/api/v1/sessions/{id}/discard-receipts`
2. Cross-reference with ingestion logs
3. **Pass Criteria:** Every rejection has corresponding receipt

---

Appendix C: Implementation Roadmap

C.1 Phase 1: Core Protocol (Months 1-3)

- Event ingestion with Ed25519 signing
- Immutable storage integration (IPFS/S3)
- Basic state machine (SYNC_WS, DEGRADED)

- REST API for cold replay

C.2 Phase 2: Hysteresis & Recovery (Months 4-6)

- HysteresisBreaker implementation
- SYNC_FV mode with deterministic recovery
- Automated monitoring and alerting

C.3 Phase 3: Audit Infrastructure (Months 7-9)

- Certified test runner
- DiscardReceipt archival system
- Audit API enhancements
- Documentation and training materials

C.4 Phase 4: Production Hardening (Months 10-12)

- Performance optimization
- Horizontal scaling architecture
- Security audit and penetration testing
- Disaster recovery procedures

Document Version: 1.0.0
Last Updated: December 20, 2025
License: CC BY-NC-SA 4.0 (Academic Research)

How to Audit This Specification

1. **Verify Invariants:** Check that all LTL formulas are precisely specified with enforcement and verification methods
2. **Threat Model Coverage:** Confirm all Byzantine scenarios are addressed
3. **State Machine Completeness:** Ensure all transitions and failure modes are explicit
4. **Cryptographic Soundness:** Validate signature scheme selection and key management
5. **Recovery Determinism:** Test that cold replay produces identical state
6. **Forensic Completeness:** Confirm no event outcome is unrecorded