

Cryptography-Driven Deterministic Logical Timeline

AI Behavior Records: Dual-Timestamp Binding and Trigger Modes

A Defensive Disclosure Technical Report

Shanghai Huapait Technology Co., Ltd.

Qiu Yong Wang Zhehao

Corresponding author: Qiu Yong

Email: qiuyong@huapait.com.cn

October 9, 2026

Defensive Disclosure Statement

This paper is a **defensive disclosure technical report**. Its purpose is to place the following three technical schemes into the public domain as prior art, preventing any entity from patenting them:

1. Request-receipt dual-timestamp binding mechanism
2. Three trigger modes for the deterministic logical timeline
3. Multi-physical-time-axis concurrent recording architecture

These three technical schemes were developed during the implementation of an AI behavior auditing system and constitute engineering enhancements to an existing deterministic logical timeline framework. This paper does not claim any patent rights and provides these schemes as publicly available prior art for free use by the industry.

Note

This is the English translation of the paper originally published in Chinese on AiraXiv (ID: 2610.0017). The research content is identical.

Abstract

An AI behavior auditing system based on a deterministic logical timeline binds cryptographic states to logical moments, making every AI decision traceable, independently verifiable, and infinitely replayable. When this system is applied to scenarios involving external large-model API calls, three engineering problems arise: an API call spans multiple logical ticks while the behavior record binds only the request moment; a single mode of logical counter advancement cannot accommodate intermittent calls and continuous monitoring; and concurrent recording across multiple physical media lacks architectural guidance. This paper proposes a request-receipt dual-timestamp binding mechanism, three trigger modes (periodic, event, and hybrid), and a multi-physical-time-axis concurrent recording architecture. It provides complete data structures, algorithm flows, auditing methods, and experimental data, as prior art for free use by the industry. This paper also explicitly delineates the security boundaries and trust assumptions of each mechanism, and provides engineering implementation schemes for authentication tag construction, clock trust models, and cross-axis causality proofs.

Keywords: cryptography-driven logical timeline; AI behavior auditing; dual-timestamp binding; trigger modes; multi-physical timeline; defensive disclosure

Contents

1	Introduction	4
1.1	Research Background	4
1.2	Three Engineering Problems	4
1.3	Contributions	5
2	Background: Deterministic Logical Timeline	5
2.1	DSF Triple	5
2.2	PSSM Evolution	5
2.3	Dual Anchoring and Behavior Chain	6
2.4	Precise Definition of Auth_Tag	6
3	Request-Receipt Dual-Timestamp Binding	6
3.1	Problem Definition	6
3.2	Dual-Timestamp Binding Mechanism	7
3.3	Comparison with Single-Timestamp Binding	7
3.4	Clock Trust Model and Tolerance Strategy	7
3.5	Timeout Event Recording	8
3.6	Auditing Method	9
3.7	Experimental Data	9
4	Three Trigger Modes	9
4.1	Problem Definition	9
4.2	Three Trigger Mode Definitions	10
4.3	Rule_ID Extension	10
4.4	Mode Switching Rules	10
4.5	Multi-Connection Arbitration Details	11
4.6	Applicable Scenario Comparison	11
4.7	Experimental Data	11
5	Multi-Physical-Time-Axis Concurrent Recording Architecture	12
5.1	Problem Definition	12
5.2	Architecture Definition	12
5.3	Recording Architecture	12
5.4	Audit Independence	13
5.5	Cross-Axis Causality Proof	13
5.6	Applicable Scenarios	14
6	Security Boundaries and Engineering Considerations	14
6.1	Threat Model and Security Boundaries	14
6.2	Key Management and Cross-Axis Consistency	15
6.3	Experimental Parameters and Reproduction Notes	15
6.4	Comparison with Existing Schemes	16
7	Conclusion	16
7.1	Contributions	16
7.2	Known Limitations and Future Work	17
7.3	Defensive Disclosure Statement	17
7.4	Relationship with Existing Patents	18

A	Symbol Table	19
B	Abbreviations	19

1 Introduction

1.1 Research Background

AI agents have rapidly penetrated critical domains such as financial trading, medical diagnosis, autonomous driving, and industrial control. These scenarios impose strict requirements on the traceability and auditability of AI behavior. Existing AI auditing schemes (log systems, blockchain attestation, TEE remote attestation) all have fundamental limitations: logs can be tampered with by administrators, blockchain attestation depends on third-party consensus, and TEE attestation reports cannot be aligned across domains.

To address these limitations, the applicant’s prior patents (Application Nos. 2026110706945, 2026111276840, etc.) proposed an AI behavior auditing scheme based on a deterministic logical timeline. The core mechanisms of this scheme include:

- **Deterministic logical timeline:** a discrete logical moment sequence composed of a logical counter n and logical state $S(n)$, decoupled from physical clocks
- **Logical-physical dual anchoring:** each logical moment n simultaneously binds the logical state hash $Hash(S(n))$ and the physical anchor $T_{anchor}(n)$
- **Behavior chain:** behavior records at each logical moment are linked through a cryptographic hash chain, forming a tamper-evident evidence chain
- **Independent verification:** the verifier holds the same DSF parameters and independently recomputes $S(n)$, without relying on any third party

The detailed mechanisms of this scheme are disclosed in Paper 1, “Cryptography-Driven Deterministic Logical Timeline: A Temporal Baseline for Machine Collaboration and AI Auditing”[9]. This paper is an engineering enhancement and defensive disclosure of Paper 1, proposing solutions to three engineering problems.

1.2 Three Engineering Problems

During the implementation of this scheme as an SDK (dsf-ai v0.1), three engineering problems were identified:

Problem 1: Characterizing the moments of cross-cycle API calls

A large-model API call (e.g., DeepSeek) takes approximately 700 ms. If the logical tick $T_{step} = 1$ ms, one call spans approximately 700 logical ticks. However, the current behavior record binds only the request moment n_{req} and cannot characterize the receipt arrival moment n_{resp} . This creates an audit blind spot: the auditor cannot verify whether the receipt was delayed.

Problem 2: The mode of logical counter advancement

The current implementation assumes that the logical counter n increments on each behavior event. This is suitable for intermittent API calls but not for continuous monitoring scenarios (e.g., autonomous driving requires recording perception data every 10 ms). Moreover, if there is no behavior for a long time, the logical timeline stalls, losing its role as a temporal baseline.

Problem 3: Concurrent recording across multiple physical media

When a node has multiple physical media (multi-NIC, multi-RF, MIMO beams), multiple business connections can truly run in parallel at the same physical moment. However, how to organize concurrent records of multiple connections and ensure the integrity of behavior chains lacks clear architectural guidance.

1.3 Contributions

This paper proposes three technical mechanisms addressing the above three problems:

1. **Request-receipt dual-timestamp binding:** behavior records simultaneously bind the request moment (n_{req}, T_{anchor_req}) and the receipt moment ($n_{resp}, T_{anchor_resp}$), supporting deterministic upper-bound verification of call latency
2. **Three trigger modes:** introducing the *Trigger_Mode* field into *Rule_ID*, defining periodic trigger, event trigger, and hybrid trigger modes
3. **Multi-physical-time-axis concurrent recording:** each physical medium corresponds to a physical time axis, and business connections bind to different axes, achieving true physical-layer parallelism

This paper provides complete data structures, algorithm flows, auditing methods, and experimental data for each mechanism, as prior art for free use by the industry. It also explicitly delineates the security boundaries and trust assumptions of each mechanism.

2 Background: Deterministic Logical Timeline

2.1 DSF Triple

The deterministic logical timeline is built on the Dynamic Security Foundation (DSF) triple:

$$DSF = (K_{sec}, Init_Anchor, Rule_ID)$$

Element	Meaning
K_{sec}	Shared secret key, pre-shared via a secure channel
$Init_Anchor$	Initial anchor, defining the logical timeline starting point and physical start anchor
$Rule_ID$	State transition rule identifier, defining the logical tick period and state evolution function

2.2 PSSM Evolution

The Protocol Security State Machine (PSSM) evolves according to the following formulas:

$$S(0) = H(K_{sec} || Init_Anchor || "INIT")$$

$$K_e = H(K_{sec} || e || "SUBKEY"), \quad e = \lfloor n / N_{epoch} \rfloor$$

$$S(n+1) = H(K_e || S(n) || "CHAIN_TAG")$$

where H is a cryptographic hash function (e.g., SHA-256), and N_{epoch} is the number of logical ticks per epoch.

2.3 Dual Anchoring and Behavior Chain

At logical moment n , the node performs the following operations:

1. Output the current logical state $S(n)$
2. Bind the deterministic behavior to $S(n)$, generating behavior record $Record(n)$
3. Instantaneously capture the local physical clock reading $T_{anchor}(n)$ as the physical anchor
4. Link $Record(n)$ to the behavior chain: $Chain_Link(n) = H(Chain_Link(n-1) || Record(n))$

2.4 Precise Definition of Auth_Tag

The authentication tag $Auth_Tag$ uses HMAC-SHA256 construction, covering all critical fields of the behavior record:

$$Auth_Tag = \text{HMAC-SHA256}(K_e, \\ n || S_{hash}(n) || Action_Type || Action_Attributes \\ || Input_Hash || Output_Hash || T_{anchor}(n) \\ || Node_ID || Chain_Link(n-1) || Tag_Version)$$

where:

- K_e is the time-bound subkey of the current epoch, derived as $K_e = H(K_{sec} || e || \text{"SUBKEY"})$, where $e = \lfloor n / N_{epoch} \rfloor$;
- $Tag_Version$ distinguishes different versions of the tag construction; the current version is 2;
- $Chain_Link(n-1)$ is the chain link value of the previous record, cryptographically binding the current record to the preceding record;
- All fields are concatenated after being encoded according to a deterministic serialization specification (e.g., RFC 8785 JSON Canonicalization).

Cross-epoch verification: When n_{req} and n_{resp} span different epochs, the verifier uses the corresponding epoch subkeys $K_{e_{req}}$ and $K_{e_{resp}}$ to recompute the $Auth_Tag$ for the two moments. If the $Auth_Tag$ in the record matches either recomputation result, the authentication for that moment passes. Both $K_{e_{req}}$ and $K_{e_{resp}}$ can be independently derived by the verifier from K_{sec} and the corresponding epoch number, without interacting with the execution node.

Relationship with Paper 1: The $Auth_Tag$ construction in this paper is consistent with that defined in Paper 1, with the only addition being coverage of the dual moments (n_{req} and n_{resp}). The $Auth_Tag$ in Paper 1 covers all fields at a single moment; this paper extends it to cover all fields at two moments.

3 Request-Receipt Dual-Timestamp Binding

3.1 Problem Definition

Consider a deterministic behavior that calls an external large-model API. The behavior initiates a request at logical moment n_{req} and receives a response at logical moment n_{resp} , spanning $\Delta n = n_{resp} - n_{req}$ logical ticks.

The existing scheme's behavior record binds only n_{req} , causing two problems:

1. **Audit blind spot:** the auditor cannot verify the receipt arrival moment. An attacker can delay the receipt, and the auditor cannot detect it
2. **Call latency unverifiable:** the auditor cannot compute the actual latency of the API call or verify whether it satisfies the business latency constraint

3.2 Dual-Timestamp Binding Mechanism

Core idea: The behavior record binds both the request moment and the receipt moment.

Definition: The request-receipt dual-timestamp binding record $Record(n_{req})$ is defined as:

$$Record(n_{req}) = \{n_{req}, T_{anchor_req}, S_{hash}(n_{req}), \\ n_{resp}, T_{anchor_resp}, S_{hash}(n_{resp}), \\ Action_Type, Input_Hash, Output_Hash, Auth_Tag\}$$

Algorithm flow:

1. **Request phase:** Before calling the external service, allocate logical moment n_{req} , sample T_{anchor_req} , and compute $S_{hash}(n_{req})$
2. **Call phase:** Call the external service and wait for the response. During this phase, the PSSM continues to evolve, with n incrementing from n_{req} to n_{resp}
3. **Receipt phase:** After receiving the response, record the current logical moment n_{resp} , sample T_{anchor_resp} , and compute $S_{hash}(n_{resp})$
4. **Record generation:** Write the four moment anchors and the input/output hashes into the behavior record

3.3 Comparison with Single-Timestamp Binding

Dimension	Single-Timestamp Binding	Dual-Timestamp Binding
Bound moments	Only request moment n_{req}	Request moment n_{req} + receipt moment n_{resp}
Call latency verification	Not verifiable	Verifiable: $\Delta T = T_{anchor_resp} - T_{anchor_req}$
Receipt delay detection	Not detectable	Detectable
Record size	Baseline	Extra 48 bytes (uncompressed)
Audit capability	Only confirms request occurred	Confirms request, receipt arrival, and actual latency

The additional overhead of dual-timestamp binding is only about 48 bytes (uncompressed) per record, but it buys two key capabilities: call latency verifiability and receipt delay detectability.

3.4 Clock Trust Model and Tolerance Strategy

Trust assumption of physical anchors: The credibility of the physical anchor T_{anchor} depends on the integrity of the local clock. This paper assumes that the local clock is short-term stable under normal operating conditions (typical crystal oscillator drift rate 1–100 ppm), but may be untrustworthy in the following scenarios:

- An attacker with system-level privileges can tamper with the local clock;
- The system clock is maliciously adjusted by an NTP service;
- Virtual machine migration causes clock jumps.

Auditor's tolerance strategy: When verifying $\Delta n \cdot T_{step} \approx \Delta T$, the auditor adopts the following tolerance strategy:

- **Absolute tolerance:** ϵ_{abs} is taken as an integer multiple of T_{step} , typically $2 \cdot T_{step}$;
- **Relative tolerance:** ϵ_{rel} is taken as 1%–5% of ΔT , to cover network jitter;
- **Statistical threshold:** The auditor maintains a historical distribution of ΔT ; when a new record's ΔT deviates from the historical distribution by more than 3σ , it is marked as anomalous.

The verification criterion is:

$$|\Delta n \cdot T_{step} - \Delta T| \leq \max(\epsilon_{abs}, \epsilon_{rel} \cdot \Delta T)$$

Multi-source time anchoring: To improve the credibility of physical anchors, a multi-source anchoring strategy is recommended. Each physical anchor is composed of multiple independent sources:

$$T_{anchor} = \{T_{local}, T_{gps}, T_{ntpsec}, T_{roughtime}\}$$

where:

- T_{local} : local system clock reading;
- T_{gps} : GPS Pulse-Per-Second (PPS) timestamp (if a GPS module is available);
- T_{ntpsec} : NTPsec authenticated time;
- $T_{roughtime}$: Roughtime network timestamp.

The auditor uses a 2-of-3 strategy to determine anchor credibility: when the readings of at least two sources differ by no more than δ (typically 10 ms), the physical anchor is considered credible. If all sources are unavailable, it degrades to the single source T_{local} , and the audit conclusion explicitly notes the degradation.

3.5 Timeout Event Recording

Design principle: The underlying recording module is only responsible for faithfully recording; timeout handling is left to the upper-layer application.

Mechanism:

1. Compute the logical span of this call: $\Delta n = n_{resp} - n_{req}$
2. Compute the physical latency of this call: $\Delta T = T_{anchor_resp} - T_{anchor_req}$
3. Compare Δn with the preset threshold $N_{timeout}$
4. If $\Delta n > N_{timeout}$, mark $Timeout : true$ in the behavior record
5. The underlying layer does not reject the record, only marks the timeout; the upper-layer application decides whether to retry, degrade, or alert based on business requirements

3.6 Auditing Method

The verifier holds the same DSF parameters and performs the following verifications:

1. **Logical binding verification:** independently recompute $S(n_{req})$ and $S(n_{resp})$, and verify consistency with the hashes in the record
2. **Temporal consistency verification:** verify $n_{resp} > n_{req}$, and $|\Delta n \cdot T_{step} - \Delta T| \leq \max(\epsilon_{abs}, \epsilon_{rel} \cdot \Delta T)$
3. **Authentication tag verification:** verify that $Auth_Tag$ covers n_{req} and n_{resp} ; for cross-epoch cases, recompute separately using $K_{e_{req}}$ and $K_{e_{resp}}$
4. **Timeout event verification:** if a timeout is marked, verify that the timeout determination conforms to the preset rules

3.7 Experimental Data

Test scenario: The Agent consecutively calls the DeepSeek API 500 times; the request-receipt logical span and physical latency of each call are recorded.

Experimental results:

Indicator	Value
Total calls	500
Average physical latency ΔT	726 ms
Average logical span Δn	726
Minimum physical latency	310 ms
Maximum physical latency	1651 ms
Latency standard deviation	189 ms
Audit pass rate	100% (500/500)
Dual-timestamp binding verification	100% pass

Conclusion: The dual-timestamp binding mechanism passed verification in 100% of 500 real calls. The auditor can precisely verify the request moment, receipt moment, and actual latency of each call.

4 Three Trigger Modes

4.1 Problem Definition

The advancement mode of the logical counter n needs to accommodate different business scenarios:

- **Intermittent API calls:** uneven call intervals, possibly from tens of milliseconds to several seconds
- **Continuous monitoring:** requires fixed-tick sampling, e.g., every 10 ms
- **Mixed scenarios:** both periodic tasks and event-driven tasks

A single advancement mode cannot cover all scenarios. Therefore, a trigger mode configuration must be introduced into $Rule_ID$.

4.2 Three Trigger Mode Definitions

Mode 1: Periodic Trigger

The logical counter n increments according to the logical tick period T_{step} defined in *Rule_ID*. The physical clock only provides a tick reference and does not participate in the n increment decision.

$$n(t) = \lfloor (t - T_{start}) / T_{step} \rfloor$$

Applicable scenarios: continuous monitoring, periodic reporting, real-time control.

Mode 2: Event Trigger

The logical counter n increments on each behavior event.

$$n_{k+1} = n_k + 1 \quad \text{if and only if a behavior event occurs}$$

Applicable scenarios: intermittent API calls, on-demand tasks, low-power IoT devices.

Mode 3: Hybrid Trigger

Primarily event-driven; when no behavior event occurs for a preset time T_{idle} , it increments periodically according to T_{step} to maintain the liveness of the logical timeline.

$$n(t) = \begin{cases} n_{last} + 1 & \text{behavior event occurs} \\ n_{last} + \lfloor (t - t_{last}) / T_{step} \rfloor & \text{idle timeout} \end{cases}$$

Applicable scenarios: composite scenarios with both periodic and event-driven tasks.

4.3 Rule_ID Extension

Add the *Trigger_Mode* field to *Rule_ID*:

$$Rule_ID = \{T_{step}, N_{epoch}, Trigger_Mode, \dots\}$$

Trigger_Mode	Value	Description
Periodic trigger	"periodic"	Increments by T_{step}
Event trigger	"event"	Increments on each behavior
Hybrid trigger	"hybrid"	Event-based, periodic keep-alive

For the hybrid trigger mode, a T_{idle} field is also required to define the idle timeout threshold.

4.4 Mode Switching Rules

The trigger mode can be switched at runtime, but the following rules must be observed:

1. **Switching timing:** switching is allowed only at logical tick boundaries
2. **State inheritance:** the current value of n is unchanged during switching; only the subsequent increment rule changes
3. **Anchor alignment:** the correspondence between the physical anchor and the logical moment for the first increment after switching is computed according to the new mode
4. **Record marking:** the switching event itself is written as a behavior record into the behavior chain, with *Action_Type* = "MODE_SWITCH"

4.5 Multi-Connection Arbitration Details

On the same physical time axis, multiple business connections share the same logical timeline. When multiple connections have behavior events at the same logical moment, arbitration is needed to avoid logical moment conflicts. This paper adopts **deterministic priority arbitration based on Rule_ID**:

1. Each connection C_i configures a priority P_i in *Rule_ID*;
2. When multiple connections compete at the same logical moment n , the connection with the highest priority (smallest P_i) is selected for execution;
3. Failed connections enter a deterministic queue and retry at the next logical moment;
4. The arbitration result for all connections is uniquely determined by the deterministic rules in *Rule_ID*, without signaling interaction.

Arbitration pseudocode:

```
for each logical tick n:
    candidates = {C_i | C_i has event at n}
    if len(candidates) > 1:
        winner = argmin(C_i.P_i for C_i in candidates)
        for C_i in candidates - {winner}:
            C_i.enqueue(n + 1)
    execute(winner, n)
```

Latency analysis: The worst-case latency introduced by arbitration is $(M-1) \cdot T_{step}$, where M is the number of concurrent connections on the same axis. With $T_{step} = 1$ ms, even with $M = 10$, the worst-case latency is only 10 ms, satisfying most real-time business requirements.

4.6 Applicable Scenario Comparison

Scenario	Recommended Mode	Rationale	Typical T_{step}
Real-time control	Periodic trigger	Fixed tick satisfies control cycle	0.125 ms
Intermittent calls	API Event trigger	No resource consumption when idle	—
Continuous monitoring	Periodic trigger	Ensures data continuity	10 ms
Mixed tasks	Hybrid trigger	Balances periodic and event	$1 \text{ ms} + T_{idle}=100 \text{ ms}$
Low-power IoT	Event trigger	Minimizes wake-up	—

4.7 Experimental Data

Test scenario: Run 100 calls in each of the three trigger modes, comparing resource usage and response latency.

Mode		Logical Advancements	Average Latency	Audit Pass Rate
Periodic	trigger	500	726 ms	100%
$(T_{step} = 1 \text{ ms})$				
Event	trigger	100	726 ms	100%
Hybrid	trigger	150	726 ms	100%
$(T_{idle}=100 \text{ ms})$				

Conclusion: All three trigger modes correctly record behavior, with a 100% audit pass rate. The event trigger mode has the fewest logical advancements in intermittent scenarios, offering the best resource efficiency.

5 Multi-Physical-Time-Axis Concurrent Recording Architecture

5.1 Problem Definition

With a single physical medium (single NIC, single RF), only one behavior can be carried at the same physical moment. Multiple business connections must be staggered or arbitrated through the logical timeline to avoid collisions.

When a node has multiple physical media (multi-NIC, multi-RF, MIMO beams), each medium corresponds to an independent physical time axis. At this point, multiple business connections can truly run in parallel at the same physical moment.

5.2 Architecture Definition

Definition: The multi-physical-time-axis concurrent recording architecture is defined as follows:

1. The node has M physical media, each corresponding to a physical time axis PT_j ($j = 1, \dots, M$)
2. Each business connection C_i is bound to a physical time axis $PT_{j(i)}$
3. Each business connection C_i has an independent DSF triple $(K_{sec,i}, Init_Anchor_i, Rule_ID_i)$
4. Business connections on different physical time axes can execute concurrently at the same physical moment
5. Multiple business connections on the same physical time axis are staggered or arbitrated through the logical timeline

5.3 Recording Architecture

Connection	Physical Medium	Logical Timeline	Behavior Chain
C_1	NIC 1	LT_1 (PSSM 1)	BC_1
C_2	NIC 2	LT_2 (PSSM 2)	BC_2
C_3	NIC 3	LT_3 (PSSM 3)	BC_3

Key design:

- Each business connection has an independent PSSM and behavior chain
- The logical moments n of different connections increment independently without mutual interference
- The physical anchors T_{anchor} of different connections are sampled independently
- The auditor verifies each behavior chain separately, without cross-connection coordination

5.4 Audit Independence

Each behavior chain can be independently verified by the auditor:

1. The verifier holds the DSF parameters of the corresponding business connection
2. Independently run the PSSM and recompute the logical state $S(n)$
3. Verify the integrity of the behavior chain
4. Use the physical anchor for cross-validation with external evidence

Advantage: An anomaly in one business connection does not affect the auditing of other connections.

5.5 Cross-Axis Causality Proof

In the multi-physical-time-axis architecture, if a logical workflow spans multiple physical media (e.g., the Agent initiates a request on NIC A and receives the result on NIC B), the independent behavior chains of each axis cannot directly prove cross-axis causality. This paper proposes two cross-axis causality proof schemes:

Scheme A: Periodic cross-axis anchoring

Each axis PT_j periodically (every T_{cross} , typically 1 second) submits its chain head hash to a global anchoring service:

$$CrossAnchorRecord_j = \{ChainHeadHash_j, T_{anchor}, PT_ID_j\}$$

The global anchoring service aggregates the *CrossAnchorRecord* of all axes into a global anchor record:

$$GlobalAnchor = Hash(CrossAnchorRecord_1 || \dots || CrossAnchorRecord_M)$$

Each axis writes the *GlobalAnchor* into its own behavior chain at the next logical moment. The auditor can detect cross-axis inconsistencies by comparing the *GlobalAnchor* of each axis.

Scheme B: Instant cross-axis reference

A cross-axis expected response placeholder is reserved in the record initiating the request:

$$Record_{req}(n_{req}) = \{\dots, CrossAxisRef : \{target_PT, expected_n_{resp}\}\}$$

A back-reference record is written in the axis receiving the response:

$$Record_{resp}(n_{resp}) = \{\dots, CrossAxisRef : \{source_PT, n_{req}, source_ChainHeadHash\}\}$$

The auditor can verify the authenticity of the cross-axis causality by comparing the *CrossAxisRef* on both sides.

Performance impact: The additional overhead of Scheme A is one global anchoring per T_{cross} , adding approximately 100 bytes of storage and one hash computation per axis per second on average; the additional overhead of Scheme B is approximately 32 bytes per cross-axis record. Neither scheme significantly increases system overhead.

Current implementation status: Both Scheme A and Scheme B are not yet implemented in the current version and belong to future work. This paper provides the design schemes for reference by the industry.

5.6 Applicable Scenarios

Scenario	Description
Multi-NIC server	Each NIC carries one business connection, achieving physical-layer parallelism
MIMO multi-beam base station	Each beam corresponds to one business connection
Multi-sensor robot	Each sensor interface corresponds to one business connection
Multi-antenna UAV	Each antenna's transmission and reception are recorded independently

6 Security Boundaries and Engineering Considerations

6.1 Threat Model and Security Boundaries

The mechanisms in this paper are built on the security model defined in Paper 1, "Cryptography-Driven Deterministic Logical Timeline". This section clarifies the security boundaries of the three engineering enhancement mechanisms.

Attacker capabilities: Consistent with Paper 1. The attacker can intercept, tamper with, replay, delay, or drop any network message; can physically capture any node; can control some nodes; but does not possess K_{sec} or the current epoch subkey K_e .

Security properties unchanged by this paper's mechanisms:

- Unforgeability: an attacker not possessing K_{sec} cannot forge a valid $S(n)$ or $Auth_Tag$;
- Independent verifiability: any verifier holding the same DSF can independently recompute $S(n)$;
- Tamper-evidence: the behavior chain uses hash-chain linking; tampering with any record breaks all subsequent links from that moment.

New security boundaries introduced by this paper's mechanisms:

Mechanism	Security Boundary
Dual-timestamp binding	Can detect receipt delay, but the credibility of the physical anchor T_{anchor} depends on the integrity of the local clock; if the local clock is tampered with, dual-timestamp binding cannot detect it. Mitigation: multi-source time anchoring (see Section 3.4)
Trigger mode switching	The switching event itself is written into the behavior chain, but there may be a one-tick boundary ambiguity during mode switching; switching is only allowed at logical tick boundaries
Multi-physical time axis	The independence of each chain is guaranteed by DSF key isolation; cross-axis causality proof is provided by Scheme A or Scheme B in Section 5.4

Uncovered attack scenarios:

- Local clock tampering: if an attacker has system-level privileges, the local physical clock can be tampered with, and the physical anchor of dual-timestamp binding becomes untrustworthy. Mitigation: multi-source anchoring (GPS + NTPsec + Roughtime) or third-party timestamp services.
- K_{sec} exposure: if the root key is exposed, the attacker can forge any record. Mitigation: K_{sec} stored in HSM/TEE, time-bound subkey layered control.
- Rollback attack: if an attacker can roll back persistent storage, the behavior chain may be truncated. Mitigation: periodically anchor the chain root hash to external trusted storage.

6.2 Key Management and Cross-Axis Consistency

Key management: The multi-physical-time-axis architecture assigns an independent DSF triple to each business connection. This raises two engineering problems:

1. **Key distribution:** each connection independently distributes DSF parameters, with complexity growing linearly with the number of connections. Hierarchical key derivation is recommended:

$$K_{sec,i} = KDF(K_{root}, connID_i || Init_Anchor_i)$$

where K_{root} is stored only in HSM/TEE and never leaves the secure boundary.

2. **Key rotation and revocation:** each connection can independently rotate its time-bound subkey K_e ; when a connection is revoked, a revocation record *RevocationRecord* is written into the behavior chain, and the auditor determines the termination moment of the chain from the revocation record.

Cross-axis consistency: Within a single physical time axis, the integrity of the behavior chain is guaranteed by the hash chain. Across physical time axes, the chains are mutually independent, and the current version does not provide cross-chain causality proof. If cross-axis association is required (e.g., the Agent initiates a request on NIC A and receives the result on NIC B), Scheme A or Scheme B in Section 5.4 can be used.

6.3 Experimental Parameters and Reproduction Notes

Test environment:

Item	Configuration
Operating system	macOS 15.7.9 (x86_64)
Python version	3.9.6
External API	DeepSeek API (cloud)
Network conditions	LAN, no artificial injected latency
Clock source	System clock (time.perf_counter_ns)

Key parameters: $T_{step} = 1$ ms, $N_{epoch} = 10^6$, $N_{timeout}$ configured per business.

Record size accounting: Dual-timestamp binding adds the following to each record compared to single-timestamp binding:

- Receipt logical moment n_{resp} : 8 bytes
- Receipt physical anchor T_{anchor_resp} : 8 bytes

- Receipt logical state hash $S_{hash}(n_{resp})$: 32 bytes (SHA-256)
- Total approximately 48 bytes (uncompressed); approximately 64 bytes if Base64-encoded

Reproduction method: The experiment script relies only on the Python standard library and can be reproduced with the following command:

```
python3 -m dsf_ai.benchmark --calls 500 --mode dual-timestamp
```

Statistical reporting: The mean, standard deviation, minimum, and maximum of the experimental data are given in Sections 3.6 and 4.7.

6.4 Comparison with Existing Schemes

The comparison of this paper’s mechanisms with existing schemes is as follows:

Scheme	Temporal Binding	Cross-Domain Verification	Extra Overhead
RFC 3161 timestamp	Single moment	Requires trust in TSA	Medium
Merkle batch anchoring	Batch	Requires trust in anchoring service	Low
Signed sequence counter	Single moment	Requires trust in signer	Low
This work’s dual-timestamp binding	Dual moment	Hold DSF	Low (48 bytes/record)

RFC 3161 timestamp: single-event timestamp service; each call requires interaction with the TSA, with high latency and bandwidth overhead; this work’s dual-timestamp binding is completed within the record, requiring no external interaction.

Merkle batch anchoring: aggregates events into a Merkle tree and periodically anchors; delay detection capability is weak (only detectable at anchoring period boundaries); this work’s dual-timestamp binding can detect receipt delay in real time.

Signed sequence counter: each record is signed by a private key and contains an incrementing sequence number; tampering can be detected but call latency cannot be verified; this work’s dual-timestamp binding additionally provides call latency verifiability.

Advantages of this work’s scheme: While maintaining low overhead (48 bytes/record), it provides three capabilities: real-time receipt delay detection, call latency verification, and cross-epoch verification, without relying on external timestamp services or trust anchors.

7 Conclusion

7.1 Contributions

This paper proposes three engineering enhancement mechanisms for an AI behavior auditing system based on a deterministic logical timeline:

1. **Request-receipt dual-timestamp binding:** behavior records simultaneously bind the request moment and receipt moment, supporting deterministic upper-bound verification of call latency. 500 real-call experiments prove a 100% audit pass rate
2. **Three trigger modes:** introducing the *Trigger_Mode* field into *Rule_ID*, supporting periodic trigger, event trigger, and hybrid trigger modes, adapting to different business scenarios

3. **Multi-physical-time-axis concurrent recording:** each physical medium corresponds to an independent time axis, and business connections bind to different axes, achieving physical-layer parallelism

This paper also explicitly delineates the security boundaries and trust assumptions of each mechanism, and provides engineering implementation schemes for authentication tag construction, clock trust models, and cross-axis causality proofs.

7.2 Known Limitations and Future Work

Known limitations:

Limitation	Description
Timeout handling in high-latency scenarios	Dual-timestamp binding only marks timeout in high-latency scenarios ($\Delta n > N_{timeout}$); the underlying layer does not reject the record; whether to retry, degrade, or alert is determined by the upper-layer application
Mode switching boundary	Trigger mode switching can only occur at logical tick boundaries; the switching event itself is written as a behavior record into the behavior chain, possibly introducing an additional logical tick of latency
Key management for multi-physical time axes	Each business connection requires an independent DSF triple, and key management complexity grows with the number of connections
Experimental scope	The experimental data in this paper comes from internal tests and has not been independently reproduced by a third party; the experiments were conducted in a LAN environment without adversarial network conditions

Future work:

1. Extend the dual-timestamp binding mechanism to multi-hop call chains (Agent $\rightarrow$ model $\rightarrow$ tool);
2. Study adaptive switching strategies for trigger modes, automatically selecting the optimal mode based on business load;
3. Implement dynamic binding and unbinding of multi-physical time axes;
4. Implement Scheme A or Scheme B for cross-axis causality proof;
5. Conduct systematic experiments under adversarial network conditions (latency jitter, packet loss, clock tampering);
6. Invite third parties to independently reproduce the experimental data in this paper.

7.3 Defensive Disclosure Statement

This paper is a **defensive disclosure technical report**. The above three mechanisms were developed during the implementation of an AI behavior auditing system and constitute engineering enhancements to an existing deterministic logical timeline framework. This paper publishes them as prior art to prevent any entity from patenting them.

This paper does not claim any patent rights, and the related technical schemes are available for free use by the industry.

7.4 Relationship with Existing Patents

The three mechanisms described in this paper are built on the applicant’s prior patents:

- Application No. 2026110706945: Deterministic behavior recording, auditing, and replay (providing the logical timeline, dual anchoring, and behavior chain foundation)
- Application No. 2026111276840: AI Agent behavior compliance control (providing safety windows and operation credential mechanisms)
- Application No. 2026101471226: Handover access patent (providing primary/sub logical timeline and timeline cluster concepts)
- Application No. 2026102177047: Multi-business scenario logical timeline collision avoidance (providing multi-business connection isolation mechanisms)

The three mechanisms in this paper are engineering detail supplements to the above patents and do not constitute new independent inventions.

AI Usage Statement

The LaTeX formatting, layout, and language polishing of this paper were assisted by AI. The research design, technical approach, core mechanisms, experimental validation, and conclusions were completed by the authors. AI did not participate in the formulation of research questions, selection of technical routes, generation of experimental data, or interpretation of results.

Shanghai Huapait Technology Co., Ltd.
October 9, 2026

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] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [3] V. Costan and S. Devadas, “Intel SGX explained,” *IACR Cryptology ePrint Archive*, vol. 2016, p. 086, 2016.
- [4] AIIR: AI Integrity Receipts, 2026.
- [5] S. A. Crosby and D. S. Wallach, “Efficient data structures for tamper-evident logging,” in *USENIX Security Symposium*, 2009, pp. 317–334.
- [6] B. Schneier and J. Kelsey, “Secure audit logs to support computer forensics,” *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 159–176, 1999.
- [7] A. Pudovkina, “Secure logging schemes: A survey,” *Journal of Computer Security*, 2020.
- [8] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, “Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP),” RFC 3161, 2001.
- [9] Qiu Yong, Wang Zhehao, “Cryptography-Driven Deterministic Logical Timeline: A Temporal Baseline for Machine Collaboration and AI Auditing,” AiraXiv, ID: 2610.0018, 2026.

A Symbol Table

Symbol	Meaning
DSF	Dynamic Security Foundation
K_{sec}	Shared secret key
K_e	Time-bound subkey
$Init_Anchor$	Initial anchor
$Rule_ID$	State transition rule identifier
$PSSM$	Protocol Security State Machine
$S(n)$	Logical state
$T_{anchor}(n)$	Physical anchor
n	Logical counter
T_{step}	Logical tick period
N_{epoch}	Number of logical ticks per epoch
$Chain_Link(n)$	Behavior chain link value
$Auth_Tag$	Authentication tag
n_{req}	Request logical moment
n_{resp}	Receipt logical moment
T_{anchor_req}	Request physical anchor
T_{anchor_resp}	Receipt physical anchor
$N_{timeout}$	Timeout threshold (logical ticks)
T_{idle}	Idle timeout threshold for hybrid trigger
$Trigger_Mode$	Trigger mode
PT_j	The j -th physical time axis
C_i	The i -th business connection

B Abbreviations

Abbreviation	Full Name
DSF	Dynamic Security Foundation
PSSM	Protocol Security State Machine
HMAC	Hash-based Message Authentication Code
HKDF	HMAC-based Key Derivation Function
HSM	Hardware Security Module
TEE	Trusted Execution Environment
TSA	Time Stamp Authority
NIC	Network Interface Card
MIMO	Multiple-Input Multiple-Output