0.1 Title of the Invention

System and Method for Falsifiable Process Evidence via Cryptographic Causality Locks and Behavioral Attestation

0.2 Technical Field

This disclosure relates to computer security, digital forensics, and authorship verification. It focuses on systems that produce tamper-evident records about how digital artifacts were created, using time-locking, device binding, and append-only evidence logs.

0.3 Background

Generative models have turned authorship disputes into an arms race. Detectors try to infer "human vs. machine" from the content itself, but any classifier can be probed and defeated by iterative generation. Certification models shift trust to third parties and then become the new point of attack. Traditional cryptographic integrity (hashing and signing) only proves that bits didn't change after signing; it doesn't say anything about how those bits came to be. When a challenger can simply say "this could have been generated," vague doubt persists.

What's missing is a process-centric record that pushes disputes into specific, testable allegations. Instead of "this might be fake," the challenger should be forced to allege a concrete mechanism (e.g., back-dating via clock rollback, device substitution, or log forgery) and a bounded time window where that mechanism occurred. That's the gap this system fills.

0.4 Brief Summary

The system produces falsifiable process evidence by combining three independent mechanisms:

1) Causality locks based on verifiable delay functions (VDFs) so checkpoints can't be silently back-dated.
2) Behavioral and hardware binding so the evidence is tied to real-time activity on a specific device (jitter/keystroke binding plus optional PUF/TPM attestations).
3) A ratcheted evidence log in a Merkle Mountain Range (MMR) so earlier checkpoints remain safe even if the device is compromised later.

It also records a Capture Environment Declaration (CED) at session start, uses evidence-scoped keys that are ratcheted and destroyed after sealing, and can optionally anchor checkpoint roots to public time sources. Evidence is exported as a portable packet with inclusion proofs, timing proofs, and optional attestations, all independently verifiable.

0.5 Brief Description of the Drawings

Figure 1 shows a high-level architecture with witnessd between applications and the file system, secure storage, evidence layers, and optional external anchors. Figure 2 shows a

commitment event from volatile buffers through WAL, CED/VDF binding, device binding, and MMR append. Figure 3 shows a VDF-locked timeline of checkpoints.

0.6 Detailed Description

0.6.1 Overview

The system runs as a background daemon ("witnessd") with a control and verification interface ("witnessctl"). The daemon monitors creation activity (e.g., document hashes, timing signals, and edits) and stores tamper-evident records in local secure storage. Evidence is exportable into a portable packet for third-party verification.

The core of the system is a forensic triad: (1) Causality Lock, (2) Behavioral-Hardware Binding, and (3) Ratcheted Evidence Log. These components are designed to be independent. A challenger who disputes the evidence must therefore allege multiple, specific failures rather than a single vague doubt.

0.6.2 Capture Environment Declaration (CED)

At session start, the system records a Capture Environment Declaration—an explicit, structured snapshot of the observable environment. Typical fields include: OS name and version, kernel version, secure boot status (if available), virtualization indicators, TPM presence and version, monotonic clock readings, wall-clock readings, process identifiers, and the hash of the witnessd executable. If a signal can't be determined, the declaration records that absence explicitly (e.g., "secure boot unknown"). The CED hash is bound into every checkpoint and evidence packet.

This matters because it prevents "they never checked X" arguments. Either the environment was as declared, or the capture mechanism was falsified; there's no middle story.

0.6.3 1. Causality Lock (VDF-based anti-back-dating)

Each checkpoint includes a VDF proof computed over a checkpoint input that binds the document hash, previous chain state, and session metadata. A VDF is sequential by design, so it forces real elapsed time. This prevents quiet insertion of back-dated checkpoints and yields a verifiable minimum elapsed time between states without relying on a trusted clock.

In practical terms, a challenger who claims back-dating must allege a concrete mechanism: either a VDF shortcut (no known attacks) or a pre-computation strategy that presupposes knowledge of future document states.

0.6.4 2. Behavioral-Hardware Binding (jitter/keystroke binding + PUF/TPM)

Behavioral binding anchors evidence in real-time activity. The system can use cryptographic keystroke binding (a jitter seal) that injects imperceptible timing offsets derived from a session secret and the evolving document hash. Those offsets form a chained proof: if you didn't have the session secret at the time, you can't reconstruct the timing sequence later. It's a process proof without recording keystroke content.

Hardware binding further ties evidence to a physical device. In one embodiment, a Physical Unclonable Function (PUF) produces a device-specific challenge/response, used to derive or bind session keys. In another embodiment, a TPM or secure enclave provides attestations and monotonic counters that show the chain wasn't rolled back. These bindings are optional and tiered; when present, they provide device-identity guarantees beyond what software alone can offer.

Keystroke Jitter Seal (Detailed Description) The jitter seal binds process activity to document evolution without capturing keystroke content. In one embodiment, the system observes an input event stream (e.g., keystroke events or application edit events) and computes a per-event timing offset derived from a session secret and the current document state. The offset is applied as a small delay (for example, microseconds to a few milliseconds) that is below the threshold of user perception but is measurable by the system.

An example construction is: J_i = HMAC(S, H_i || c_i || t_i || J_{i-1}) mod R where S is a session secret, H_i is a hash of the document or process state at event i, c_i is the event ordinal, t_i is a timestamp or monotonic counter, J_{i-1} is the prior jitter value, and R is a configured jitter range. The resulting jitter value J_i is applied to the event timing and recorded (or derivable) as part of the checkpoint evidence.

This construction has three properties: (1) it binds the timing sequence to the document state, so post-hoc generation would require all intermediate states; (2) it chains each event to the previous event, creating a tamper-evident timing ledger; and (3) it requires the session secret at the time of creation, making offline synthesis without key compromise infeasible. Verification recomputes the expected jitter values from the evidence record and checks them against the observed or recorded timing sequence. Because only timing and hashed state are used, no keystroke content is stored.

0.6.5 3. Ratcheted Evidence Log (Merkle Mountain Range)

Checkpoints are appended to a Merkle Mountain Range. Each leaf contains the checkpoint hash plus bound metadata (including the CED hash and device-binding data when present). The MMR root (bagged peaks) is the authenticated state. Because each append deterministically changes the root, any removal or modification of past entries is detectable.

The log is ratcheted forward: each checkpoint is signed with a scoped session key, and the key is advanced and destroyed after use. That's important in a post-compromise scenario. Even if an attacker gets the current key, they can't rewrite older checkpoints without breaking the MMR or faking past signatures.

0.6.6 Evidence-Scoped Keys and Lifecycle

The system uses per-session or per-artifact keys derived via HKDF from a device-bound or master identity secret. It records key lifecycle metadata: when the key was generated, first used, last used, and destroyed. That lifecycle is bound to the evidence packet.

This avoids the classic "the key could've been stolen anytime" argument. Instead, the

challenger has to allege a specific window where key compromise occurred and show a trace consistent with that window.

0.6.7 Evidence Production Flow

A representative flow looks like this:

(a) Capture signals (document hashes, timing signals) into a volatile buffer.
(b) Periodically seal entries into a durable write-ahead log (WAL).
(c) On a heartbeat or semantic trigger (e.g., save), aggregate a checkpoint snapshot.
(d) Bind the CED hash and compute the VDF proof (causality lock).
(e) Apply device binding (PUF response and/or TPM attestation), then sign with the scoped key.
(f) Append the checkpoint hash to the MMR and record the new root.
(g) Ratchet and destroy the session key.

0.6.8 Optional External Anchoring

For higher assurance, the system can anchor MMR roots to public time sources (e.g., public ledgers or trusted timestamp authorities). Anchoring doesn't define trust, but it constrains time claims and makes clock manipulation arguments harder to sustain.

0.6.9 Verification and Export

An evidence packet includes: MMR inclusion proofs, VDF parameters and outputs, CED data, optional device attestations, and optional process declarations. A verifier checks:

- Chain integrity (hashes + MMR proofs)
- Minimum elapsed time (VDF verification)
- Device binding (PUF/TPM artifacts if present)
- Key lifecycle bounds

If any invariant fails, verification records the failure and, if configured, downgrades the evidence class. The goal isn't to claim perfect truth; it's to make challenges specific and testable.

0.6.10 Tiered Evidence Model

Evidence tiers add independent components based on risk tolerance:

- Basic: MMR chain + process declaration
- Standard: add behavioral binding (jitter seal / presence)
- Enhanced: add PUF + VDF
- Maximum: add external anchors

Each tier raises the number of independent allegations needed to dispute a record, which is the core of adversarial collapse.

0.6.11 Example Use Cases

The system applies to authorship disputes, compliance logging, software supply-chain provenance, and scientific reproducibility. In each case, the evidence isn't about the aesthetics of the content; it's about a defensible record of the process by which the content evolved.

0.7 Scope and Variations (Non-Limiting)

The embodiments described above are illustrative. The invention isn't limited to any particular cryptographic primitive, hardware module, or logging structure so long as the system produces falsifiable process evidence by binding time, device identity (or device-specific signals), and append-only integrity into a unified record.

0.7.1 Time-Locking Alternatives

The causality lock can be implemented with any mechanism that provably or practically enforces minimum elapsed time between checkpoints, including but not limited to: verifiable delay functions, sequential proof-of-work, chained hash delays, trusted hardware clocks with monotonic counters, or periodic external anchors that bound time. The key requirement is that a verifier can establish a minimum time separation between checkpoints or a non-back-dateable ordering.

0.7.2 Device-Binding Alternatives

Device binding can be implemented with any mechanism that ties the evidence record to a specific physical or logical device, including but not limited to: hardware PUFs, software-simulated PUFs, TPM or secure enclave attestation, device-unique key derivation, hardware-backed key stores, or a composite of independent device fingerprints. Behavioral binding can be implemented with any cryptographic signal derived from real-time interaction, such as keystroke timing, pointer dynamics, sensor timing, or application event timing, provided the signal is chained to document state or process state.

0.7.3 Append-Only Log Alternatives

The ratcheted evidence log can use any append-only authenticated data structure, including but not limited to: Merkle Mountain Range, Merkle trees with periodic roots, history trees, hash-chained logs, or append-only ledger systems. The essential property is that modifications or deletions of prior entries are detectable by verification.

0.7.4 Key-Lifecycle Alternatives

Evidence-scoped keys can be generated per session, per artifact, or per commit. Key-lifecycle declarations may be implicit (derivable from timestamps and signatures) or explicit (recorded lifecycle fields). Forward secrecy can be achieved by ratcheting, one-way key evolution, or destructive key handling.

0.7.5 Evidence Tiering and Optional Components

Any component may be optional or tiered depending on threat model. In a minimal configuration, the system may provide a time-locked append-only chain with process declarations. In higher-assurance configurations, device binding, behavioral binding, and external anchors are added. The independence of these components is a feature: it forces any dispute to allege multiple specific failures.

0.7.6 Application Breadth

The system applies to any digital artifact or process where falsifiable process evidence is valuable, including but not limited to: authorship disputes, compliance logging, software build provenance, scientific data collection, design workflows, and regulatory reporting.