The Cryptographic Imperative: Hardening Ternary Moral Logic with a Ternary Merkle Architecture

Lev Goukassian
Architect of Ternary Moral Logic
Santa Monica, California

ORCID: 0009-0006-5966-1243

February 14, 2026

Interactive Version: *This document is available as a live web artifact containing dynamic structural formatting.*

Abstract

This research establishes the cryptographic imperative for Ternary Moral Logic (TML) as a mechanism for enforceable AI governance. While traditional AI safety relies on post-hoc auditing, TML enforces a "No Log = No Action" architecture where execution is cryptographically bound to an immutable record. We analyze the information-theoretic properties of a Ternary (Base-3) Merkle tree topology, demonstrating that it aligns semantically with TML's triadic logic (+1, 0, -1) while reducing Merkle inclusion proof size by approximately 35% compared to standard binary structures at scale ($N=10^9$). Furthermore, we introduce the **Active Axiom Set Hash**, a novel binding mechanism that cryptographically freezes the governing ethical ruleset at the precise moment of inference. This mechanism renders "retroactive reinterpretation"—the falsification of historical rule application—computationally infeasible. A comparative analysis against Bitcoin, Ethereum, and Certificate Transparency logs validates the TML architecture's superior suitability for high-frequency, privacy-preserving, and intergenerationally durable ethical enforcement.

Keywords: AI Governance, Ternary Merkle Trees, Cryptographic Auditability, Active Axiom Set Hash, Ternary Moral Logic.

Foundational Role of the Merkle Tree in Ethical Guarantees

The Ternary Moral Logic (TML) framework necessitates a structural substrate capable of providing cryptographically verifiable, immutable, and context-aware evidence of AI-driven deliberations [168, 169]. The Merkle tree architecture is not merely an implementation detail but the indispensable foundation upon which TML's core ethical guarantees are built. Its removal would catastrophically undermine the system's ability to provide legally defensible and intergenerationally durable records of moral decisions. The necessity of the Merkle tree stems from its unique capacity to bind discrete pieces of information into a single, provable commitment, enforce contextual integrity through cryptographic hashing, and create a tamper-evident historical chain resistant to truncation and retroactive alteration [37, 107]. Without this structure, TML would lack a formal mechanism to freeze outcomes, anchor decisions to their governing ruleset, or enable efficient, lightweight verification by third parties, rendering its ethical claims unverifiable and therefore functionally void.

The primary function of the Merkle tree in TML is to serve as a cryptographic seal for the system's history. Each leaf node commits to a specific moral event, and the aggregation of these commitments into a hierarchical structure produces a single root hash that represents the entire state of the log at a given point in time [41]. This root hash is then anchored to a public, append-only ledger or trusted timestamping service, creating a permanent and publicly verifiable record [60, 94]. The critical property here is collision resistance; it is computationally infeasible to find two different sets of data that produce the same root hash [274]. This ensures that any change to a single leaf node—the representation of a moral event—will propagate up the tree, altering every parent hash until the root itself is changed. Consequently, proving the integrity of any individual event requires only a small, logarithmic-sized set of sibling hashes, known as a Merkle proof, which allows for efficient verification without needing access to the entire dataset [42, 165]. If the Merkle tree were replaced with a simple hash chain or an unstructured append-only log, verifying the existence and content of an event would require inspecting and reprocessing all preceding events, a process that is inefficient and scales poorly. The Merkle tree is therefore necessary because it transforms a raw sequence of data points into a structured, efficiently verifiable history.

A second, more sophisticated role of the Merkle tree is in enforcing contextual integrity through the "Active Axiom Set Hash." Each leaf node must contain a cryptographic hash of the exact rule-set active at the moment the moral event was processed [173]. This design choice is fundamental to preventing retroactive reinterpretation of past decisions, a significant adversarial threat. An attacker attempting to alter historical decisions by changing the underlying axioms would first need to generate a new, valid Axiom Set Hash. However, this new hash would not correspond to the hash committed to the leaf node during the original event's processing. Any verifier using a Merkle proof for that historical event would attempt to reconstruct the path to the root using the modified axioms, resulting in a mismatch with the originally anchored root hash. This discrepancy cryptographically proves the attempted tampering. The integrity of the historical record is thus intrinsically tied to the context (the active rules) in which it was created. Removing the Merkle structure would sever this binding, making it impossible to bundle an event's payload with its governing context into a single, self-contained, and verifiable unit. One could store the event data and the rule-set separately, but there would be no efficient, cryptographic method to prove they co-existed at a specific point in time without transmitting both in full, defeating the purpose of lightweight verification.

The concept of "freezing Sacred Zero outcomes" as an immutable moral state is another pillar of TML made possible by the Merkle tree. A Sacred Zero event signifies a decision to take no action or a prohibition against an action, and its status must be permanent and irrevocable. In a standard logging system, entries can be deleted or overwritten. Within a Merkle tree structure, however, the leaf containing the Sacred Zero event becomes part of a larger cryptographic commitment. Altering the content of this leaf to change the outcome would constitute a second preimage attack—that is, finding a new input that produces the same hash output as the original leaf hash—a problem considered computationally infeasible for secure hash functions [141]. Furthermore, since the leaf is part of a tree whose root is anchored to a public record, its presence and content are permanently bound to a specific point in time. The only way to remove the commitment would be to break the hash function itself or to control the entire system to perform a catastrophic truncation, both of which are outside the assumed threat model for a properly secured system. The Merkle tree provides a strong guarantee that once a Sacred Zero is recorded and its root is anchored, its existence and nature are frozen in the cryptographic ledger of the system's history. The collapse of this guarantee would occur if the Merkle tree were removed, leaving only a database of mutable records that could be silently altered or deleted by an insider with sufficient privileges.

Finally, the architectural design supports a concrete scenario demonstrating how the Merkle tree prevents retroactive reinterpretation. Consider a loan application decision logged as a TML event. The leaf node contains several key components:

1. The hash of the event payload (`Hash(EventData)`), which includes the applicant's details, credit score, and the decision outcome.
2. The hash of the active axiom set at that time (`Hash(AxiomSet_v1.2)`).

3. The hash of the canonical schema used to serialize the event (`Hash(Schema_v3.1)`).

These hashes form the foundational leaves of the tree. For instance, `LeafHash = SHA256(Hash(EventData) || Hash(AxiomSet_v1.2) || Hash(Schema_v3.1))`. This `LeafHash` is then incorporated into the Merkle tree structure. An attacker attempting to retroactively reinterpret the decision a year later by modifying `AxiomSet_v1.2` to `AxiomSet_v2.0` faces an insurmountable challenge. To make the change plausible, they would need to compute a new `Hash(AxiomSet_v2.0)`. When a verifier requests a Merkle proof for the original event, the proof will include the sibling nodes required to traverse from the leaf up to the master root. During verification, the verifier recomputes this path using the provided event payload and the newly computed `Hash(AxiomSet_v2.0)`. This recomputed path will inevitably lead to a different root hash than the one that was previously anchored to a public blockchain. This mismatch is immediate and undeniable proof of tampering. The Merkle tree structure is what makes this detection both efficient and universally verifiable, forming the bedrock of TML's trustworthiness.

Canonical Leaf Node Specification and Privacy-Preserving Serialization

The integrity of the entire Ternary Moral Logic (TML) system hinges on the rigorous and deterministic specification of its fundamental unit: the canonical leaf node. This node serves as the immutable container for a single moral event, encapsulating not just the event's data but also the essential metadata required to verify its authenticity, context, and legality at any point in the future. The design mandates a strict schema, deterministic serialization, and robust privacy-preserving measures to ensure that every leaf is cryptographically sound, unambiguous, and compliant with data protection principles. The canonical schema is explicitly defined to include a set of mandatory fields, each serving a distinct purpose in building a complete and verifiable record. These fields are: Event ID, Monotonic Sequence ID, Trusted Timestamp, Sacred Zero trigger source, Pillar reference, Risk classification, Reflection outcome, Integrity flags, Schema Version ID, Schema Hash Commitment, Active Axiom Set Hash, and Hash Algorithm Version ID [173].

The Event ID is a globally unique identifier, typically generated using a standard like UUID, which ensures that no two events can share the same identity, thereby preventing ambiguity and enabling precise referencing. The Monotonic Sequence ID provides an ordered index for events, which is crucial for establishing temporal ordering and detecting gaps or replays in the log. The Trusted Timestamp is a cryptographically signed timestamp from a reliable source, binding the event to a specific moment in time and protecting against backdating or future-dating attacks. The remaining fields collectively capture the semantic content and context of the moral decision. The Sacred Zero trigger source identifies the origin of a non-action decision, the Pillar reference categorizes the event under one of the core ethical pillars (Human Rights, Earth Protection, Governance), and the Risk classification and Reflection outcome detail

the reasoning behind the decision. Integrity flags are reserved for system-level metadata, such as proof-of-storage attestations or encryption status. The final four fields are critical for cryptographic integrity: the Schema Version ID and its corresponding Schema Hash Commitment ensure that the structure of the event data is part of the immutable record, preventing silent schema evolution attacks. The Active Axiom Set Hash binds the event to the exact rules that governed it at the time of creation, as previously discussed. Finally, the Hash Algorithm Version ID specifies which cryptographic hash function was used to create the leaf's internal hashes, a vital field for long-term cryptographic survivability [20].

To ensure that the Merkle tree construction is deterministic and reproducible, the serialization format for the canonical schema must be strictly defined. All fields must be serialized in a fixed, predictable order. Locale independence is enforced by mandating standardized formats for dates, numbers, and strings (e.g., UTF-8 encoding). Non-deterministic values, such as random number generator outputs or system-specific timestamps without a fixed base, must be explicitly rejected during the leaf creation process. The use of a strict, canonical serialization standard, such as CBOR or Protocol Buffers, is recommended to eliminate any ambiguity in how the data is converted into a byte string before hashing [185]. This determinism is paramount; any variation in the serialization process would result in different hash values for semantically identical events, breaking the cryptographic chain of custody and invalidating all subsequent proofs. The leaf node itself does not store the raw event data but rather its cryptographic hash, ensuring that even if the underlying storage is compromised, the integrity of the leaf's commitment remains intact.

Privacy is a central tenet of the TML architecture, and the leaf specification incorporates multiple layers of protection. Raw personal data is never hashed or stored within the TML system. Before hashing, all personally identifiable information (PII) must undergo irreversible pseudonymization or redaction [164]. Pseudonymization involves replacing identifiers with non-reversible tokens, while redaction removes the data entirely from the payload destined for the Merkle tree. The redaction process itself must be auditable, with a detailed trace logged showing which data elements were removed and the logic applied. This approach aligns with privacy-by-design principles and regulations like GDPR [80, 87]. By committing only anonymized or redacted data to the public-facing Merkle tree, the system preserves the privacy of individuals involved in moral events while still maintaining a complete and verifiable record of the event's outcome and context. The actual raw data, which may contain sensitive information, is stored separately in an encrypted, private repository linked via a reference from the canonical payload, accessible only under strict regulatory authority and cryptographic controls [127].

Immutability is enforced at the leaf level through the combination of schema versioning and hash commitments. The Schema Version ID and Schema Hash Commitment within each leaf create a tight coupling between the event's data and its structure. Any modification to the schema definition—adding, removing, or changing a field—necessitates a new Schema Version ID. Consequently, any attempt to apply an old schema to new data will be detected during validation, as the committed schema hash will not match the hash of the data serialized under

the new schema. This prevents schema evolution attacks where an adversary might try to introduce a new field to carry hidden data or manipulate existing data structures. If a mutation occurs to any field within a leaf node, the hash of that field's value will change, invalidating the entire path up the Merkle tree to the root. This cascading effect ensures that any unauthorized modification is immediately detectable by any party possessing the correct root hash and a valid Merkle proof. The combination of a deterministic serialization format, strict privacy controls, and schema versioning creates a leaf node that is cryptographically sealed, contextually aware, and resilient to tampering and unauthorized modification.

Field Name	Type	Description
`EventID`	Globally Unique Identifier	A unique ID for the event, e.g., a UUID. Prevents collisions. [214]
`SequenceID`	Monotonically Increasing Integer	A sequential index for ordering events chronologically. Detects replays and gaps. [46]
`Timestamp`	ISO 8601 String	A trusted, cryptographically signed timestamp. Binds event to a point in time. [60]
`TriggerSource`	String	Identifies the entity or condition that initiated a 'Sacred Zero' outcome. [168]
`PillarRef`	Enum ('Human Rights', 'Earth Protection', 'Governance')	Categorizes the event under a core ethical domain. [86]
`RiskClass`	String	Classification of the risk associated with the event. [173]
`Outcome`	String	The reflection outcome of the moral deliberation. [173]
`IntegrityFlags`	Bitfield	System-level flags for metadata like proof-of-storage status. [48]
`SchemaVersionID`	String/Integer	Version identifier for the canonical schema used.

Field Name	Type	Description
		Enables schema governance. [173]
`SchemaHashCommitment`	Hash (SHA-3-256)	Cryptographic hash of the schema definition corresponding to `SchemaVersionID`. [173]
`AxiomSetHash`	Hash (SHA-3-256)	Cryptographic hash of the active rule-set at the time of the event. Enforces contextual integrity. [173]
`HashVersionID`	String/Enum	Identifier for the hash algorithm used (e.g., 'SHA3-256', 'PQ-HASH-A'). Critical for post-quantum migration. [20]

Ternary Tree Construction Model and Hierarchical Integrity Enforcement

The selection of a ternary Merkle tree topology over the more conventional binary model is a deliberate architectural decision driven primarily by performance optimization, with semantic interpretation layered on top [91, 147]. While a binary tree offers simplicity, a ternary tree provides structural advantages in terms of reduced height, smaller proof sizes, and lower computational overhead for a given number of leaf nodes. For a balanced binary tree with n leaves, the height is $\log_2(n)$, whereas for a balanced ternary tree, the height is $\log_3(n)$. Since $\log_3(n)$ is always less than $\log_2(n)$ for $n > 1$, the ternary structure is inherently shallower. This reduction in height has direct consequences for the efficiency of the system. Smaller proof sizes mean that verifiers require fewer hashes to confirm an event's inclusion, reducing bandwidth consumption and storage costs. Lower computational overhead translates to faster root calculations and proof verifications, which is critical for meeting stringent latency requirements [280, 282]. The user's directive to prioritize performance justifies this complexity, especially when considering modern hardware capabilities like SIMD instructions that can accelerate computations on higher-fanout data structures [217, 275].

The mathematical modeling of the tree's geometry confirms these benefits. For a system processing 1,000,000 events, a binary tree would have a height of approximately 19.93, requiring up to 20 hashes in a Merkle proof. A ternary tree of the same size would have a height of approximately 12.59, requiring only 13 hashes in a proof. This represents a 35% reduction in

proof size, a significant saving for a system designed to support lightweight third-party verification. The analysis of CPU and memory overhead shows a similar trend; traversing fewer nodes to reach the root from a leaf reduces the computational load on both client verifiers and the system generating the proofs [282]. The visual comparison of binary versus ternary trees illustrates this geometric advantage clearly: the ternary tree achieves the same coverage with a more compact, vertically shorter structure. This performance gain is not merely academic; it directly impacts the system's scalability and its ability to operate within the sub-2-millisecond inference latency constraint. The semantic mapping of the ternary states (e.g., Moral, Immoral, Sacred Zero) to the three branches of the tree is flexible and not dogmatic, allowing the logical interpretation to evolve independently of the underlying physical structure [91].

The construction of the tree must adhere to strict requirements to maintain its integrity and performance characteristics. The tree is built asynchronously to avoid blocking inference operations, utilizing a rolling buffer that holds new events and maintains an integrity checksum until a batch can be processed . Leaf nodes are placed deterministically based on their monotonic sequence ID, ensuring a consistent and predictable layout. Special handling is required for scenarios with an odd number of leaves at any level, which can be managed by duplicating the rightmost leaf or using a placeholder null hash. Crucially, the tree must be kept balanced to preserve the logarithmic depth and, therefore, the efficiency of proofs. Balancing algorithms, such as those inspired by AVL or B-tree rotations, must be implemented to manage insertions and maintain optimal structure [279]. Replay protection is enforced at the leaf level by validating the uniqueness of the `EventID` and ensuring the `SequenceID` strictly increases, rejecting any out-of-order or duplicate entries [214]. This combination of asynchronous construction, deterministic placement, balancing, and replay protection ensures that the Merkle tree remains a robust and efficient data structure under heavy load.

To organize the vast volume of moral events, the TML architecture employs a hierarchical integrity model with separate subtrees for distinct ethical domains: Human Rights, Earth Protection, and Governance [86]. This segmentation improves audit clarity by logically partitioning the data, allowing verifiers to focus on relevant domains without being overwhelmed by unrelated events [86]. Each subtree is itself a Merkle tree, with its own root hash. These subtree roots are then aggregated into a higher-order Merkle tree, culminating in a single master root that represents the entirety of the TML log. This multi-level structure provides both granular and holistic views of the system's state. The master root can be anchored to a public ledger, providing a single point of truth for the entire system's history. Simultaneously, the subtree roots can be anchored independently or at different frequencies, offering flexibility in auditing and governance. For example, the Human Rights subtree might be subject to monthly regulatory audits, while the overall master root is anchored quarterly.

Forward integrity is a cornerstone of the hierarchical model, ensuring that the history cannot be altered without detection. This is achieved through a strict forward hash chain. Each new root hash in the hierarchy must cryptographically include the hash of the previous root. For instance, when a new block of events is added to the Human Rights subtree, the new subtree root `R_new`

is calculated, and the system anchors a record containing both `R_new` and the previous root hash `R_prev`. This creates an unbroken chain of custody: `NewAnchor = SHA256(R_new || R_prev)`. Any attempt to modify a past event would require recalculating all subsequent subtree and master roots, a task that is computationally infeasible. The root-of-roots versioning scheme ensures that the entire hierarchical structure is treated as a single, evolving artifact, with each anchoring event creating a new version of the complete history [18, 19]. This forward integrity guarantee means that upon intrusion, an attacker cannot counterfeit logging data generated before their system control was established, preserving the trustworthiness of the historical record [19].

Adversarial Hardening and Data Availability Resilience

The TML architecture is engineered to withstand a comprehensive adversarial threat model that includes malicious insiders, external attackers, and systemic failures [14, 43]. Every component of the system is hardened to mitigate specific threats, from data at rest to data in transit and during processing. A malicious insider with write access to the log poses a significant risk, as they could attempt to alter or delete historical events. The Merkle tree structure inherently thwarts this by making any leaf modification computationally infeasible without a second preimage attack [141]. To prevent truncation, the forward integrity mechanism, where each root includes the hash of the prior root, creates a continuous chain that would be broken by any deletion of historical blocks [19]. The causal ordering proof and execution interlock further protect against a developer or insider attempting to bypass the Sacred Zero check by ensuring that inference outputs are cryptographically tied to the committed state of the log [130, 207]. Similarly, a developer attempting a silent schema modification is thwarted by the requirement to increment the `SchemaVersionID` and commit the new schema hash to the next leaf, making such changes auditable and transparent [173].

External threats, such as network interception and storage compromise, are addressed through a defense-in-depth strategy. While an attacker intercepting network traffic could see the Merkle proofs and event payloads, they cannot forge a valid proof without access to the corresponding leaf data or the ability to break the cryptographic hash function [274]. In the event of storage compromise, the encryption keys for the raw event data are ephemeral and stored in hardware-backed secure enclaves, minimizing the window of opportunity for an attacker to decrypt and misuse the data . Even if an attacker gains temporary access to partial decryption keys, the data remains protected by short-lived session keys [100]. The system is also resilient to replay and truncation attacks. Replay protection is enforced at the leaf level through the uniqueness of the `EventID` and strict validation of the monotonic `SequenceID` [214]. Truncation is prevented by the forward-integrity guarantees, which ensure that the entire history up to a certain point is chained together, making any gap in the sequence easily detectable [18]. The reconciliation protocol is designed to automatically signal anomalies when missing root intervals are detected, triggering a forensic investigation .

The most critical aspect of adversarial hardening is ensuring long-term data availability. The explicit design axiom states that "a Merkle root without retrievable data fails TML governance guarantees" . This elevates data availability from a mere feature to a core requirement for the system's validity. To counteract data loss events or storage failures, the architecture mandates a redundant, geographically distributed storage model . A comparative analysis of storage approaches reveals a clear trade-off between centralized and decentralized models. Centralized models offer simplicity and potentially higher performance but introduce a single point of failure and trust, making them vulnerable to censorship and corruption [24]. Decentralized models, such as those built on protocols like Arweave or Filecoin, provide superior resilience through redundancy and censorship resistance but come with greater operational complexity and potential cost variability . An illustrative evaluation suggests that a hybrid approach may be optimal, combining the reliability of a geo-distributed cloud storage provider with the permanence guarantees of a decentralized network like Arweave, which uses a novel economic model to incentivize long-term data retention .

To bridge the gap between the cryptographic hashes in the Merkle tree and the actual data stored in the availability layer, a proof-of-storage or data availability attestation mechanism is essential. Simply having the hashes is insufficient if the data they point to is lost or inaccessible. Mechanisms like those proposed in Coded Merkle Trees (CMT) offer constant-cost protection against data availability attacks by using erasure coding, where the original data can be reconstructed from a subset of the available pieces [85, 216]. Other approaches involve periodic challenges issued to storage providers to prove they possess the data corresponding to specific hashes. For the TML system, implementing a proof-of-storage mechanism is non-negotiable. It provides the cryptographic assurance that the data referenced by the Merkle tree is not just existent but also retrievable, fulfilling the promise of a verifiable and durable record. The disaster recovery protocol must be rigorously tested to ensure that in the event of a catastrophic failure at one geographic location, the data can be fully rehydrated and the system restored to a consistent state, preserving the integrity of the entire moral history . This robust data availability strategy, combined with cryptographic hardening, ensures the system's resilience against the full spectrum of specified adversarial threats.

Lightweight Verification, Time-Bound Anchoring, and Causal Integrity

The TML architecture is designed to enable efficient, lightweight verification by regulators and other third parties, a capability fundamentally derived from the properties of the Merkle tree. This functionality mirrors the Simplified Payment Verification (SPV) model used in cryptocurrencies, allowing a verifier to confirm the inclusion of a specific event in the log without downloading or storing the entire dataset [165]. To verify a single event, a regulator needs only three pieces of information: the canonical payload of the event in question, the Merkle inclusion proof for that event, and a reference to the anchored root hash of the relevant subtree (e.g., the Human Rights subtree) [42]. The verifier then independently computes the root hash from the

payload and the proof. If the recomputed root matches the anchored root, the event is authenticated and confirmed to be part of the official record. This process is logarithmic in complexity, meaning the amount of data required for verification grows slowly even as the total number of events in the log expands. The proof itself consists of a small set of hashes corresponding to the sibling nodes along the path from the target leaf to the root, making it compact and easy to transmit and validate on resource-constrained devices like laptops or mobile phones [165]. In the event of a verification failure, the system must provide a clear error code and, if possible, a diagnosis of the cause, such as a mismatched root or an invalid proof structure, initiating a more detailed forensic review .

Ensuring time-bound integrity is critical for the legal and ethical validity of TML's records. The system must enforce a maximum anchoring delay to guarantee that the log's state is periodically and reliably committed to a public, immutable ledger . The requirement for ≤2 ms inference latency is exceptionally demanding and pushes the limits of current technology, implying a highly optimized, parallelized construction process is necessary [23, 275]. To meet this target, the system employs asynchronous tree building and rolling buffers with integrity checksums, ensuring that the computational work of building the tree does not block the primary inference pipeline . The automatic anchoring trigger mechanism initiates the anchoring process based on a timer or event threshold, ensuring regular cadence. To prevent a malicious infrastructure operator from deliberately delaying anchoring, a multi-chain anchoring strategy is employed, distributing the anchoring tasks across multiple independent blockchain networks, thereby removing reliance on any single party or network . The anchored root is accompanied by a trusted timestamp, binding the cryptographic commitment to a specific point in time and preventing backdating attacks [60]. A reconciliation protocol is in place to detect and log any missing root intervals, providing an independent audit pathway for regulators to investigate discrepancies in the anchoring schedule .

Causal integrity is paramount for enforcing the "Sacred Zero" protection mechanism. The system must provide a formal guarantee that the commitment of a Sacred Zero outcome to the log occurs before or atomically with the generation of any inference output that relies on that commitment. This is achieved through a mechanism called the Execution Interlock. The output of the AI's inference engine must cryptographically reference the hash of the committed log state. Specifically, the inference output payload must include the root hash of the Merkle tree *after* the Sacred Zero event has been appended and processed. If a system component attempts to bypass the log and generate an inference output without a corresponding entry in the Sacred Zero-comitted log, the output will lack the correct root hash reference. Verifiers of the inference output will immediately reject it as invalid, as it fails to satisfy the causal dependency on the authoritative log state [130, 207]. This interlock creates a formal barrier against silent bypasses, ensuring that no decision is ever made without its full ethical deliberation and commitment being permanently recorded. The atomic snapshot boundary defines the precise moment when the log state is finalized and its hash is propagated to dependent processes, cementing the causal link between the moral decision and its downstream consequences.

The security of the entire system rests on robust key management practices. Ephemeral encryption keys are used to protect the raw event data at rest, significantly reducing the risk of long-term data exposure . These keys should be generated, stored, and destroyed within secure, hardware-backed environments like Trusted Platform Modules (TPMs) or Hardware Security Modules (HSMs) [100]. A strict key rotation schedule, perhaps aligned with the anchoring frequency, ensures that compromised keys have a limited lifespan. A key compromise detection protocol must be implemented to monitor for signs of unauthorized key usage or leakage. Crypto-shredding is the final security measure, allowing for the controlled destruction of decryption keys. When crypto-shredding is triggered, the keys are securely erased, rendering the associated encrypted data permanently inaccessible. Critically, this process preserves the integrity of the Merkle tree; the hashes of the now-unrecoverable data remain valid and continue to form a valid part of the cryptographic chain, ensuring that the historical record of the *event* is preserved even if the raw data is gone . This combination of lightweight verification, time-bound anchoring, causal integrity enforcement, and secure key management forms a comprehensive security framework for the TML system.

Formal Integrity Guarantees and Comparative Architectural Analysis

The TML architecture derives its formal integrity guarantees from the well-established properties of the underlying cryptographic primitives, primarily the chosen hash function. The system's security model rests on several key assumptions: the collision resistance of the hash function, meaning it is computationally infeasible to find two different inputs that produce the same hash output; the preimage resistance, meaning given a hash value `h`, it is computationally infeasible to find any message `m` such that `hash(m) = h`; and the second-preimage resistance, meaning given an input m_1, it is computationally infeasible to find a different input m_2 such that $hash(m_1) = hash(m_2)$ [141, 274]. As long as these properties hold, the Merkle tree provides a robust guarantee of data integrity. Forward integrity is formally defined as the property that, after a system intrusion, an attacker who gains control of the system cannot counterfeit the logging data that was generated before their intrusion occurred [19, 20]. This is achieved in TML through the chaining of root hashes, where each new root cryptographically binds to the previous one, creating an unbreakable chain of custody [18]. The guarantees degrade only if the underlying cryptographic assumptions are broken, either through a mathematical breakthrough or the advent of a sufficiently powerful quantum computer [16].

Long-term survivability is a critical requirement for a system intended for intergenerational review. The architecture is designed to be resilient against long-term cryptographic degradation through a clear migration path for hash algorithms. The inclusion of a `Hash Algorithm Version ID` in every leaf node is a crucial design feature that enables this transition [20]. When a new, more secure hash function becomes necessary (e.g., due to advances in quantum computing), the system can begin using it for new leaves without invalidating the historical record. Historical data remains verifiable because each leaf contains the exact algorithm version

used at the time of its creation [11]. Validity-preserving overlays can be used to keep signed records verifiable through the post-quantum transition without the need to re-sign the entire historical dataset [11]. The system's reliance on hash-based signatures like XMSS or LMS further reinforces its post-quantum readiness, as these schemes are specifically designed to be resistant to attacks from quantum computers [7, 179]. While the primary role of the Merkle tree in TML is data integrity, the architectural patterns borrowed from post-quantum signature schemes provide a strong foundation for future-proofing the system's cryptographic core [52].

A comparative analysis of the TML architecture against existing systems highlights its unique position regarding the specified trade-offs: governance robustness > audit clarity > scalability. The table below summarizes this analysis.

Feature	**TML Architecture**	**Bitcoin Transaction Logs**	**Ethereum State Tries**	**Certificate Transparency (CT) Logs**
Primary Purpose	Immutable record of AI moral deliberations [168]	Secure peer-to-peer digital currency transactions	Decentralized smart contract platform	Public audit of SSL/TLS certificates [120]
Structure	Hierarchical Ternary Merkle Trees [91]	Binary Merkle Trees [108]	Modified Patricia Tries (MPT) [230]	Binary Hash Trees [128]
Governance Robustness	High. Designed for regulated environments, with features like redaction and selective disclosure [230]	Low. Permissionless, open participation [220]	Medium. Governed by community consensus, susceptible to DAO exploits [143]	Low. Open submission, but operators can censor submissions [247]
Audit Clarity	High. Hierarchical structure segments data by ethical domain, improving focus for auditors [86]	Medium. Transactions are opaque; complex to trace funds [300]	Medium. Stateful; requires understanding of account balances and storage [143]	High. Log is a simple, append-only list of certificate entries [128]

Feature	TML Architecture	Bitcoin Transaction Logs	Ethereum State Tries	Certificate Transparency (CT) Logs
Scalability	High. Ternary tree reduces proof size; parallel construction aims for high throughput [282]	Low. Block size limits transaction rate [300]	Medium. Ongoing scaling efforts (sharding, rollups) needed [62]	High. Designed for high-volume certificate submissions [128]
Key Differentiator	**Governance-first design**, integrating ethics, privacy, and verifiability [168]	**Decentralized money**	**Programmable world computer**	**Public-key infrastructure transparency**

This comparison demonstrates that while other systems leverage Merkle trees for integrity, TML is uniquely architected for governance robustness. The hierarchical ternary structure prioritizes audit clarity and efficiency, and the focus on lightweight verification addresses scalability concerns common in permissionless blockchains. The final report must conclude with a formal disclosure of failure modes and residual risks. The guarantees provided by the TML architecture are conditional. They assume the correctness of the cryptographic algorithms, the absence of a breakthrough in quantum computing, and the proper implementation of the software. Key residual risks include the compromise of hardware security modules containing signing keys, collusion between malicious insiders and external attackers, and the degradation of physical storage media over time [43, 154, 177]. A thorough understanding of these dependencies and risks is essential for deploying and maintaining the system responsibly .

Document Control

Attribute	Detail
Document ID	TML-RES-001
Version	1.0.0
Status	**PUBLISHED**

Attribute	Detail
Classification	**Cryptographic Research**
Target	**SSRN / IEEE**
Author	Lev Goukassian
Date	2026-02-14