On the Incompleteness of Cryptographically Verifiable Audit Logs Under Omission Adversaries

Tokachi Kamimura
VeritasChain Standards Organization
kamimura@veritaschain.org

December 2025

Abstract

We prove that *completeness*—ensuring all generated events are logged—is information-theoretically impossible to achieve cryptographically without external trust. This result holds even against computationally unbounded verifiers.

While *integrity* (logged data cannot be modified) is achievable through hash chains, Merkle trees, and digital signatures, completeness requires fundamentally different mechanisms. No cryptographic construction can force a malicious generator to log events it chooses to omit, because omitted events leave no cryptographic trace.

We characterize the minimal trust required: a single trusted third party suffices, and in multi-party settings, exactly $\lceil (n+1)/2 \rceil$ honest observers are necessary and sufficient. We further show that zero-knowledge proofs cannot circumvent this barrier—ZK proves correct processing of logged events but cannot prove absence of omitted events.

Keywords: Cryptographic audit trails, Impossibility theorems, Omission adversaries, Trust assumptions, Zero-knowledge proofs

1 Introduction

Cryptographic audit trails are fundamental to accountability in automated systems. From financial trading platforms to autonomous vehicles, regulators increasingly require that system decisions be logged in a tamper-evident manner. The cryptographic community has developed sophisticated techniques for ensuring *integrity*—the property that logged events cannot be modified or reordered without detection. Hash chains, Merkle trees [1], and digital signatures provide strong integrity guarantees under well-established cryptographic assumptions.

However, integrity alone is insufficient for meaningful auditability. A malicious system can maintain a perfectly intact, cryptographically secured log while systematically omitting events that would reveal misconduct. This attack—which we term an *omission attack*—cannot be detected by examining the log alone, no matter how sophisticated the cryptographic protections.

1.1 The Completeness Problem

We formalize *completeness* as the property that all events generated by a system appear in its audit log. Surprisingly, despite the extensive literature on authenticated data structures and verifiable logging, the achievability of completeness has received little formal treatment.

Consider an AI-driven trading system generating millions of decision events daily. Existing legal requirements mandate comprehensive logging for post-hoc analysis. But what prevents the system from "forgetting" to log evidence of market manipulation?

The intuitive answer—"nothing"—turns out to be formally correct and has deep implications. We prove that **completeness is impossible to achieve through cryptography alone** when the log generator is the sole participant. This is not a limitation of current techniques but a fundamental barrier analogous to the FLP impossibility result [3] for distributed consensus.

1.2 Our Contributions

We make the following contributions:

(1) **Formal Model** (Section 2): We provide rigorous definitions of audit systems, distinguishing between *integrity*, *completeness*, and *consistency*. We introduce the *omission adversary* model that captures realistic threats.

(2) **Impossibility Theorem** (Section 3): We prove that no cryptographic protocol can achieve completeness in a single-generator setting. The result holds unconditionally (information-theoretically) and does not depend on computational assumptions.

(3) **Minimal Trust Characterization** (Section 4): We characterize the trust assumptions required to achieve completeness:

   - A single trusted third party (TTP) is sufficient.
   - In a multi-party setting without a TTP, we prove that $\lceil (n+1)/2 \rceil$ honest participants are necessary and sufficient, establishing a tight bound.

(4) **Zero-Knowledge Analysis** (Section 5): We analyze whether zero-knowledge proofs can reduce trust requirements. We show that ZK proofs provide orthogonal guarantees (correct processing) but cannot overcome the completeness barrier.

(5) **Implications** (Section 6): We discuss implications for audit system design, identifying common misconceptions and providing guidance for practitioners.

Novelty and Relationship to Prior Work. While our impossibility result is conceptually related to classic results like the FLP theorem [3], we establish a *distinct* and *stronger* barrier. FLP proves that deterministic consensus is impossible in asynchronous systems due to *temporal uncertainty*—the inability to distinguish a crashed process from a slow one. Our result shows that completeness is impossible due to *informational absence*—omitted events leave no cryptographic trace.

This distinction has important consequences:

- FLP can be circumvented with randomization, partial synchrony, or failure detectors. Our barrier *cannot* be circumvented by computational assumptions or timing guarantees—it requires adding external observers.
- FLP applies to multi-party consensus; our result applies even to a *single* generator with no faults in the communication model.
- FLP is about agreement on ordering; our result is about existence verification.

Furthermore, while prior work on verifiable logging (Crosby & Wallach [6]) explicitly excludes omission adversaries, we are the first to *formalize* why this exclusion is necessary and to *characterize* the minimal trust required to achieve completeness. Our tight bound of $\lceil (n+1)/2 \rceil$ honest observers is new and provides concrete guidance for system designers.

1.3 Related Work

Authenticated Data Structures. Merkle trees [1] and their variants provide integrity for static and dynamic datasets. Certificate Transparency (CT) [2] uses append-only Merkle trees for PKI auditing. CT achieves completeness through a specific trust model: Certificate Authorities (CAs) *must* submit certificates to public logs, and monitors/auditors continuously watch for omissions. This is precisely our TTP model (Section 4): the CA is the generator, and public log servers plus monitors form the trusted observers. CT's completeness guarantee is only as strong as the assumption that monitors will detect and report missing certificates—an assumption that has been validated in practice but relies on external incentives (browser policies, reputation).

Verifiable Logging. Crosby and Wallach [6] formalized tamper-evident logging with untrusted loggers. Their model assumes the logger receives all events—precisely the assumption we question. They acknowledge this limitation: their threat model explicitly excludes adversaries who "simply do not log certain events." Our work formalizes why this exclusion is necessary and characterizes the trust required to lift it. Ma and Tsudik [7] study forward-secure audit logs but similarly assume completeness.

Byzantine Consensus. The FLP impossibility [3] shows that deterministic consensus is impossible in asynchronous systems with even one faulty process. Dwork, Lynch, and Stockmeyer [4] achieve consensus under partial synchrony. Castro and Liskov [5] provide practical Byzantine fault tolerance (PBFT) achieving consensus with $3f + 1$ nodes against f Byzantine faults. Our impossibility result is conceptually related but addresses a different problem: we show that *even a single malicious generator* can defeat completeness, regardless of synchrony assumptions.

Blockchain and Distributed Ledgers. Blockchain systems [8, 9] achieve completeness for *submitted* transactions through consensus among miners/validators. This is our multi-party model: miners are observers, and the $> 50\%$ honest mining power assumption corresponds to our honest majority requirement. However, blockchain completeness applies only to transactions broadcast to the network—transactions a user chooses not to broadcast are not covered. This distinction (observable vs. internal events, Definition 2.3) is fundamental and applies equally to blockchain systems. Additionally, blockchain's latency (minutes to hours for finality) and throughput constraints make it unsuitable for high-frequency logging applications requiring sub-second confirmation.

Accountability and Non-Repudiation. Küsters et al. [10] formalize accountability in security protocols. Their focus is on attributing blame after violations, not preventing omissions. Our work complements theirs by characterizing when omissions are detectable.

Positioning of This Work. We position our contribution as follows: Crosby and Wallach [6] solved the *integrity* problem—ensuring logged data cannot be tampered with. We address the *completeness* problem—ensuring all events are logged in the first place. While FLP [3] established impossibility due to *temporal* uncertainty (asynchrony), we establish impossibility due to *informational* absence. Our barrier is strictly stronger: it holds even in synchronous settings and against computationally unbounded verifiers.

2 System Model and Definitions

2.1 Audit Systems

Definition 2.1 (Event). *An* event $e = (id, t, payload)$ *consists of:*

- *$id \in \{0,1\}^*$: unique identifier*
- *$t \in \mathbb{N}$: timestamp (logical or physical)*
- *$payload \in \{0,1\}^*$: application-specific data*

Definition 2.2 (Event Generator). *An* event generator *$\mathcal{G}$ is a (possibly probabilistic) process that produces a sequence of events $E = (e_1, e_2, \ldots)$ over time. We write $E_{\mathcal{G}}(T)$ for the set of events generated by $\mathcal{G}$ up to time T.*

Definition 2.3 (Internal vs. Observable Events). *We distinguish two classes of events based on external visibility:*

- Internal events*: Events that exist only within the generator's computational state and are never transmitted externally.*
- Observable events*: Events that the generator transmits (or commits to) via an external communication channel.*

Formal definition. *Let $E^{\mathsf{int}}_{\mathcal{G}}(T) \subseteq E_{\mathcal{G}}(T)$ denote internal events and $E^{\mathsf{obs}}_{\mathcal{G}}(T) = E_{\mathcal{G}}(T) \setminus E^{\mathsf{int}}_{\mathcal{G}}(T)$ denote observable events. An event e is* observable *iff the generator invokes $\mathsf{Send}(e, \mathcal{P})$ for some external party $\mathcal{P}$.*

Operational criteria. *In practice, an event e is observable if any of the following holds:*

1. *e is transmitted over a network to at least one external recipient.*
2. *A cryptographic commitment $c = \mathsf{Com}(e; r)$ is transmitted externally.*
3. *e triggers an externally visible side effect (e.g., a trade execution reported to a counterparty).*

Boundary cases. *Events that are cached, buffered, or queued but not yet transmitted remain internal until transmission occurs. Partial transmission (e.g., sending to some but not all intended recipients) makes an event observable—the adversary cannot "un-send" once any external party receives it.*

This distinction is fundamental: ***internal events are information-theoretically invisible to any external verifier****, while observable events leave traces that external parties can potentially record. Our impossibility results (Section 3) apply to* all *events; our achievability results (Section 4) apply only to* observable *events. This is not a limitation of our analysis but a fundamental barrier: no mechanism can verify the existence of information that never leaves the generator.*

Definition 2.4 (Audit Log). *An* audit log *L is an ordered sequence of events with associated cryptographic metadata. We write $L[i]$ for the i-th entry and $|L|$ for the length.*

Definition 2.5 (Audit System). *An* audit system *$\Pi = (\mathsf{Setup}, \mathsf{Log}, \mathsf{Verify})$ consists of:*

- *$\mathsf{Setup}(1^\lambda) \to pp$: Generate public parameters*
- *$\mathsf{Log}(pp, L, e) \to L'$: Append event e to log L*
- *$\mathsf{Verify}(pp, L) \to \{0,1\}$: Verify log integrity*

2.2 Security Properties

Definition 2.6 (Integrity). *An audit system* Π *provides* integrity *if for all* PPT *adversaries* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda) \\ (L, L') \leftarrow \mathcal{A}(pp) \\ \mathsf{Verify}(pp, L) = 1 \wedge \mathsf{Verify}(pp, L') = 1 \\ \wedge\ L \neq L' \wedge \mathsf{diverge}(L, L') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{diverge}(L, L')$ *holds iff neither* L *is a prefix of* L' *nor* L' *is a prefix of* L*—that is, the logs have diverged (forked) at some point rather than one being a simple extension of the other.*

Integrity ensures that an adversary cannot produce two different valid logs that have forked from a common history—any modification or inconsistency is detectable.

Definition 2.7 (Completeness). *Let* Π *be an audit system with generator* $\mathcal{G}$*. Let* $E_{\mathcal{G}}(T)$ *denote the (random) set of events generated up to time* T*, and* L_T *denote the log at time* T*. The system provides* ϵ-completeness *if:*

$$\Pr\left[\, \exists\, e \in E_{\mathcal{G}}(T) : e \notin L_T \,\right] \leq \epsilon$$

Equivalently, the probability that the log fails to contain all generated events is at most ϵ*:*

$$\Pr\left[\, E_{\mathcal{G}}(T) \not\subseteq L_T \,\right] \leq \epsilon$$

The probability is over the randomness of the generator, the logging protocol, and any adversarial choices. When $\epsilon = 0$*, we say the system provides* perfect completeness*.*

Definition 2.8 (Consistency). *Let* $\{L_1, \ldots, L_k\}$ *be logs maintained by* k *parties observing the same generator. The logs are* consistent *if for all* i, j*: either* L_i *is a prefix of* L_j *or vice versa.*

2.3 Adversary Models

Definition 2.9 (Omission Adversary). *An* omission adversary $\mathcal{A}_{\mathsf{omit}}$ *controls the event generator* $\mathcal{G}$ *and can:*

1. *Generate arbitrary events*
2. *Choose which events to include in the log* L
3. *Produce valid cryptographic proofs for included events*

The adversary wins if it can produce a valid log L *such that* $E_{\mathcal{G}}(T) \not\subseteq L$ *and no verifier detects the omission.*

Remark 2.10. *The omission adversary is strictly weaker than a Byzantine adversary. It does not attempt to forge signatures, find hash collisions, or break cryptographic assumptions. It simply exercises its power to* not *log certain events.*

Definition 2.11 (External Observer). *An* external observer $\mathcal{O}$ *is a party distinct from the generator that receives information about events. We distinguish:*

- Trusted observer*: Receives all events in real-time and is assumed honest.*
- Untrusted observer*: May collude with the generator.*

Definition 2.12 (Verifier's View). *In a single-generator audit system, the* verifier's view View_V *consists of:*

1. *Public parameters* pp *from* Setup

2. *The audit log L provided by the generator*

3. *Any auxiliary information* aux *output by the generator*

Crucially, the verifier has no direct access *to:*

- *The generator's internal state or computation*
- *Events that were generated but not logged*
- *Any "ground truth" about what events should exist*

The verifier can only reason about events through their representation in L. This definition is central to our impossibility result: completeness is unverifiable precisely because omitted events leave no trace in View_V.

3 Impossibility of Cryptographic Completeness

We now state and prove our main impossibility result.

3.1 The Single-Generator Setting

Theorem 3.1 (Impossibility of Completeness). *Let Π be any audit system where the event generator $\mathcal{G}$ is the sole participant (no external observers). Then Π cannot achieve ϵ-completeness for any $\epsilon < 1$, regardless of the cryptographic mechanisms employed.*

Proof. We prove by construction. Let $\Pi = (\mathsf{Setup}, \mathsf{Log}, \mathsf{Verify})$ be an arbitrary audit system. Consider an omission adversary $\mathcal{A}_{\mathsf{omit}}$ that:

1. Generates events $E = \{e_1, \ldots, e_n\}$ according to some application logic.
2. Selects a subset $S \subset E$ to omit.
3. Constructs log L containing only events in $E \setminus S$.
4. Executes Log correctly for all events in $E \setminus S$.

We claim that $\mathsf{Verify}(pp, L) = 1$ with probability 1.
Proof of claim: The Verify algorithm can only check properties of L itself:

- Hash chain validity: All hashes are correctly computed over *logged* events.
- Signature validity: All signatures are valid for *logged* events.
- Merkle proof validity: All proofs are valid for *logged* events.
- Temporal ordering: Timestamps in L are monotonic.

Since $\mathcal{A}_{\mathsf{omit}}$ honestly executes the logging protocol for events it chooses to include, all these checks pass. Crucially, Verify has no information about events in S—they were never processed by any cryptographic operation.

Information-theoretic argument: Let $\mathcal{I}(L)$ denote the information content of log L. By construction:

$$\mathcal{I}(L) = \mathcal{I}(E \setminus S)$$

The information about S is simply not present in L. No verification algorithm, regardless of computational power, can detect the absence of information that was never recorded.

Formal entropy bound: Let $H(\cdot)$ denote Shannon entropy. The verifier's view $\mathsf{View}_V = (pp, L)$ contains no information about omitted events:

$$H(S \mid \mathsf{View}_V) = H(S)$$

That is, conditioning on the verifier's view does not reduce uncertainty about the omitted set S. This follows because S is statistically independent of View_V—the generator can choose any S and produce a valid View_V for the remaining events. No detection strategy can achieve advantage better than random guessing:

$$\Pr[\text{Verifier detects omission}] = \Pr[\text{Verifier detects omission} \mid S = \emptyset]$$

This is not a computational limitation but an *information-theoretic* one. Even an unbounded verifier cannot distinguish between:

- World 1: Generator produced events $E \setminus S$ (no omission).
- World 2: Generator produced events E but omitted S.

Both worlds produce identical logs, and the verifier's view is identical in both. Therefore, no verification strategy can achieve better than random guessing for omission detection.

Since the adversary can always omit at least one event while producing a valid log, completeness is impossible. □

3.2 Strengthening the Result

The impossibility of Theorem 3.1 is robust to various "countermeasures" that practitioners might consider. We now show that the impossibility persists even with:

1. Cryptographic commitments to event counts (Corollary 3.2)
2. Trusted timestamping authorities (Corollary 3.3)
3. Rate-based monitoring (Corollary 3.4)
4. Trusted Execution Environments (Corollary 3.5)

Each corollary follows the same proof structure: we show that the proposed mechanism operates on *data controlled by the adversary.* The adversary can provide consistent data that passes all verification checks while still omitting events. The fundamental barrier remains: external verification of completeness requires external information about what events should exist.

Corollary 3.2 (Impossibility with Commitments)**.** *The impossibility holds even if the generator must commit to the total number of events before generating them.*

Proof. We show this by construction.

Attack: Let the true event set be $E = \{e_1, \ldots, e_n\}$ and let $S \subset E$ be the set to omit. The adversary commits to $n' = n - |S|$ before generating any events. For the verifier's view, the adversary produces:

- Commitment $c = \mathsf{Com}(n'; r)$ at time t_0.
- Log L containing exactly $n' = |E \setminus S|$ events.

Verification: The verifier checks $|L| = n'$ and $\mathsf{Open}(c, n', r) = 1$. Both checks pass because the adversary committed to the *post-omission* count.

Information-theoretic argument: The commitment c binds the adversary to some value n', but n' is chosen by the adversary. The verifier has no way to know what the "correct" value should be. Formally, $H(|E| \mid c, L) = H(|E|)$ because the commitment reveals nothing about the true event count. □

Corollary 3.3 (Impossibility with Timestamps). *The impossibility holds even with trusted timestamping of logged events.*

Proof. We formalize the limitation of timestamp authorities.

Model: A Trusted Timestamp Authority (TSA) provides function $\mathsf{TS} : \{0, 1\}^* \to \mathsf{Sig}$ that, on input m, produces a signed timestamp $\sigma = \mathsf{Sign}_{sk_{TSA}}(m\|t)$ where t is the current time.

Attack: For event e_j to be omitted, the adversary simply never submits e_j to the TSA. The TSA produces no output for inputs it never receives.

Formal argument: Let $E = \{e_1, \ldots, e_n\}$ be the true event set and $S \subset E$ the omitted set. The TSA produces timestamps $\{\sigma_i : e_i \in E \setminus S\}$. The verifier sees: $\{(e_i, \sigma_i) : e_i \in E \setminus S\}$.

The verifier can verify:

- Each σ_i is valid: $\mathsf{Verify}_{pk_{TSA}}(\sigma_i, e_i\|t_i) = 1$ ✓
- Temporal ordering: $t_1 < t_2 < \cdots$ ✓

The verifier *cannot* verify:

- Whether additional events exist that were never timestamped ✗

The TSA certifies what it receives, not what should have been sent. This is the input problem: the TSA is an oracle for timestamp generation, but oracles cannot verify completeness of their inputs. □

Corollary 3.4 (Impossibility with Rate Commitments). *The impossibility holds even if the generator commits to an expected event rate.*

Proof. **Model:** Suppose the generator commits to an expected rate λ events per unit time, with variance σ^2. After time T, the expected count is $\mu = \lambda T$ with standard deviation $\sigma_T = \sigma\sqrt{T}$.

Attack: Real systems exhibit rate variability. The adversary claims a "low activity period" to justify omissions. If the observed count is $n' = n - k$ (where k events were omitted), the adversary attributes the shortfall to natural variation.

Distinguishing problem: The verifier must distinguish:

- H_0: True rate was lower; n' events is the correct count.
- H_1: True rate was λ; k events were omitted.

If $k < 2\sigma_T$ (within two standard deviations), the observation is statistically consistent with H_0. The verifier cannot reject H_0 with high confidence.

For arbitrary k, the adversary can claim a rate decrease (e.g., "market was closed") that the verifier cannot independently verify. Without external observation of actual events, rate-based claims are unfalsifiable. □

Corollary 3.5 (Impossibility with Trusted Execution Environments). *The impossibility holds even if the generator runs inside a Trusted Execution Environment (TEE) such as Intel SGX or ARM TrustZone.*

Proof. A TEE guarantees that code executes correctly and that its internal state cannot be observed or tampered with by the host. However, a TEE cannot force the host to *invoke* it. The omission adversary simply does not call the TEE's logging function for events it wishes to suppress.

More formally, TEEs solve the *oracle problem* (ensuring computation is performed correctly) but not the *input problem* (ensuring all inputs are provided). A TEE can guarantee "if event e is submitted, it will be logged correctly," but cannot guarantee "event e will be submitted." The decision to submit remains with the adversarial host.

This is analogous to the blockchain oracle problem [14]: smart contracts execute correctly, but cannot verify that off-chain data provided to them is complete or accurate. □

Remark 3.6 (TEE Extensions and Their Limits)**.** *Modern TEE deployments often include additional mechanisms. We analyze why each fails to achieve completeness:*

- ***Remote attestation****: Attestation proves that a specific enclave is running trusted code. It does not prove that the enclave has received all events—the host controls which events reach the enclave.*
- ***Sealed storage****: Sealed storage protects data confidentiality and integrity across enclave restarts. However, the adversary can simply not seal events it wishes to omit.*
- ***Monotonic counters (TPM/HSM)****: Hardware counters can prevent replay attacks and detect rollback of the enclave state. However, they cannot detect* omission*: if an event is never processed, no counter is incremented, and no detectable state change occurs.*
- ***Trusted timestamping****: A TEE can request timestamps from a trusted source (e.g., Intel's EPID attestation includes a timestamp). This proves* when *the enclave was invoked but not* whether *all events were submitted.*

The fundamental issue is that all these mechanisms operate on data inside *the TEE. The adversary's control lies* outside*—in deciding what data enters the TEE in the first place. No TEE mechanism can reach outside its boundary to verify completeness of external inputs.*

3.3 Relation to FLP Impossibility

Our result is conceptually related to the FLP impossibility theorem [3], which shows that deterministic consensus is impossible in asynchronous systems with even one faulty process.

Proposition 3.7 (Analogy to FLP)**.** *The completeness impossibility can be viewed as a degenerate case of consensus impossibility where:*

- *The "consensus" is on the set of events that occurred.*
- *There is only one "process" (the generator).*
- *The generator is "faulty" (adversarial with respect to completeness).*

However, there are important differences:

1. **Nature of impossibility**: FLP relies on asynchrony and timing uncertainty. Our result holds even in synchronous settings—the barrier is informational, not temporal.
2. **Fault model**: FLP considers crash faults or Byzantine faults. Our omission adversary is weaker—it simply chooses not to act.
3. **Circumvention**: FLP can be circumvented with randomization or partial synchrony. Our impossibility requires adding participants (observers).

Theorem 3.8 (Fundamental Distinction)**.** *Completeness impossibility is strictly stronger than consensus impossibility in the following sense: adding computational assumptions (e.g., random oracles, trusted setup) does not help overcome completeness impossibility, whereas it can help with consensus (via randomized protocols).*

Proof. Consensus impossibility in FLP arises from timing uncertainty in determining whether a process has crashed or is merely slow. Randomization [11] or partial synchrony [4] can break the symmetry that FLP exploits.

Completeness impossibility arises from *information absence.* No cryptographic primitive can generate information about events that were never processed. Even a random oracle cannot produce a proof of an event's existence if the event was never queried. Trusted setup cannot help because the adversary controls what enters the setup. □

4 Minimal Trust Assumptions for Completeness

Given that completeness is impossible without external trust, we characterize the *minimal* trust required.

4.1 Single Trusted Third Party

Theorem 4.1 (TTP Sufficiency)**.** *A single trusted third party $\mathcal{T}$ that receives all events in real-time is sufficient to achieve perfect completeness.*

Proof. Protocol:

1. Generator sends each event e_i to $\mathcal{T}$ immediately upon generation.
2. $\mathcal{T}$ maintains its own log $L_{\mathcal{T}}$ and issues signed receipt $r_i = \mathsf{Sign}_{sk_{\mathcal{T}}}(e_i, t_i)$.
3. Generator includes r_i in its log.
4. Verifier checks: for each event in generator's log, a valid receipt exists.

Completeness argument: If generator omits event e_j, then $e_j \in L_{\mathcal{T}}$ but $e_j \notin L$. The verifier can query $\mathcal{T}$ and detect the discrepancy.

Trust assumption: $\mathcal{T}$ is honest (does not collude with generator to suppress receipts). □

Remark 4.2. *This is the model used in institutional settings: trading venues report to central authorities in real-time. The authority serves as the TTP.*

4.2 Multi-Party Setting

When no single TTP is available, we consider multiple parties that collectively ensure completeness.

Remark 4.3 (Scope of Multi-Party Completeness)**.** *Per Definition 2.3, we distinguish internal events (never transmitted) from observable events (transmitted externally). The single-generator impossibility (Theorem 3.1) applies to* all *events, including internal ones. The multi-party achievability results in this section apply only to* observable *events.*

This is not circular reasoning but a fundamental information-theoretic constraint:

- *Internal events: No external party ever receives any information about these events. By definition, they are unverifiable—this is Theorem 3.1.*
- *Observable events: The generator transmits these to the network. External observers can record them, creating redundant copies that enable verification.*

The multi-party setting does not "avoid" the impossibility; rather, it operates in a regime where external information flow exists. The generator's decision to transmit an event is itself the act that makes completeness achievable—but only for that event. Events the generator chooses not to transmit remain subject to the impossibility theorem.

Remark 4.4 (Delivery Assumption)**.** *Our multi-party model assumes* reliable broadcast*: when the generator sends an observable event e to the network, all honest observers eventually receive e. This is a standard assumption in distributed systems (achievable via gossip protocols or reliable broadcast primitives). Under this assumption, if the generator transmits e, then at least $\lceil (n+1)/2 \rceil$ parties (the honest observers) will have a record of e. Attacks where the adversary prevents delivery to* all *honest observers are outside our model—such attacks constitute a network-level denial of service rather than an omission attack. The key distinction: an omission adversary chooses* not to send *certain events; it does not have the power to intercept messages sent to honest parties.*

In practice, network partitions or delays may cause some honest observers to temporarily miss events. Our results extend to partial synchrony *models [4]: if eventual delivery is guaranteed within a known bound Δ, completeness can be verified after waiting Δ time units. Fully asynchronous networks with unbounded delays require additional mechanisms (e.g., optimistic protocols with challenge periods) and are left to future work.*

Definition 4.5 (Multi-Party Audit System)**.** *A* multi-party audit system *consists of:*

- *One generator $\mathcal{G}$*
- *n observers $\mathcal{O}_1, \ldots, \mathcal{O}_n$*
- *Communication channels between all parties*

Each observer maintains its own view of events received from $\mathcal{G}$. We say the system achieves completeness if all observable *events (per Remark 4.3) appear in the final log.*

Definition 4.6 (Honest Observer)**.** *An observer $\mathcal{O}_i$ is* honest *if:*

1. *It accurately records all events it receives.*
2. *It does not collude with $\mathcal{G}$ or other corrupted observers.*
3. *It responds truthfully to verifier queries.*

Theorem 4.7 (Multi-Party Lower Bound)**.** *In a multi-party audit system with n observers, at least $\lceil (n+1)/2 \rceil$ honest observers are necessary to achieve completeness against an omission adversary that can corrupt up to $\lfloor (n-1)/2 \rfloor$ observers.*

Proof. We prove this via an indistinguishability argument.

Setup: Suppose only $h < \lceil (n+1)/2 \rceil$ observers are honest. Then $c = n - h \geq \lceil n/2 \rceil$ observers are corrupted. The adversary controls the generator plus these c observers.

Attack construction:

1. Generator creates event e^* internally.
2. Generator sends e^* to all n observers (as required by protocol).
3. Honest observers (h of them) record e^*.
4. Corrupted observers delete e^* and sign statements: "I never received e^*."
5. Generator omits e^* from log L.

Information-theoretic analysis: Define two worlds:

- **World** W_0: Generator produces events $E \setminus \{e^*\}$; honest minority falsely claims e^* exists.
- **World** W_1: Generator produces events E; corrupted majority denies e^* exists.

Let View_V denote the verifier's view (all messages, signatures, and attestations received). We show View_V is identically distributed in both worlds.

In W_0: The verifier sees $c + 1 > n/2$ parties (generator + corrupted observers) consistently attesting "e^* never existed," and $h < n/2$ parties claiming otherwise.

In W_1: The verifier sees $c + 1 > n/2$ parties (generator + corrupted observers) consistently attesting "e^* never existed," and $h < n/2$ parties claiming e^* exists.

Statistical indistinguishability: Since the adversary controls a majority, it can simulate either world perfectly. The corrupted parties produce valid signatures on their false statements (they possess the signing keys). The verifier cannot cryptographically distinguish honest from corrupted attestations—both are syntactically valid.

Formally, let D be any (possibly unbounded) distinguisher. Define the advantage:

$$\mathsf{Adv}_D = \left|\Pr[D(\mathsf{View}_V^{W_0}) = 1] - \Pr[D(\mathsf{View}_V^{W_1}) = 1]\right|$$

Without prior knowledge of which parties are honest, $\mathsf{Adv}_D = 0$ because:

1. Both worlds produce syntactically identical message distributions.
2. The only difference is semantic (which parties are lying), which is not reflected in View_V.

Decision-theoretic consequence: The verifier must choose between H_0 (no omission, minority are liars) and H_1 (omission occurred, majority are colluding). Any decision rule that does not assume prior knowledge of honest parties achieves detection probability:

$$\Pr[\text{detect omission}] \leq \frac{1}{2}$$

Standard approaches (majority voting, threshold signatures) favor H_0, so the omission succeeds. □

Theorem 4.8 (Multi-Party Upper Bound)**.** $\lceil (n+1)/2 \rceil$ *honest observers are sufficient to achieve completeness for observable events.*

Proof. We construct a protocol and prove its correctness.

Protocol Π_{obs}:

1. **Broadcast phase**: Generator broadcasts each observable event e_i to all n observers.
2. **Acknowledgment phase**: Each observer $\mathcal{O}_j$ that receives e_i signs an acknowledgment: $a_{i,j} = \mathsf{Sign}_{sk_j}(H(e_i)\|i\|j)$ where H is a collision-resistant hash.
3. **Confirmation rule**: Event e_i is *confirmed* iff $\geq \lceil (n+1)/2 \rceil$ valid acknowledgments $\{a_{i,j}\}$ exist.
4. **Verification**: Verifier accepts log L iff every confirmed event appears in L.

Security analysis:

Completeness (no false negatives): Let $e_i \in E_{\mathcal{G}}^{\mathsf{obs}}(T)$ be an observable event. By definition of observable (Definition 2.3), the generator broadcasts e_i. Under reliable broadcast (Remark 4.4), all $\lceil (n + 1)/2 \rceil$ honest observers receive e_i. Each honest observer produces a valid acknowledgment. Thus, e_i reaches the confirmation threshold. If the generator omits e_i from log L, the verifier detects the mismatch: confirmed but not logged.

Soundness (no false positives): Suppose the adversary attempts to confirm a "phantom" event e^* that was never broadcast. To reach the threshold, the adversary needs $\lceil (n+1)/2 \rceil$ acknowledgments. Honest observers never sign acknowledgments for events they did not receive. The adversary controls at most $t = \lfloor (n-1)/2 \rfloor$ observers.

Formal security reduction: We reduce soundness to the EUF-CMA (Existential Unforgeability under Chosen Message Attack) security of the signature scheme.

Let $\epsilon_{\mathsf{forge}}$ be the probability that an adversary forges a signature for any honest observer. To confirm phantom event e^*, the adversary needs at least $\lceil (n+1)/2 \rceil - t = 1$ forged signature from an honest observer. By a union bound over the $h = n - t \geq \lceil (n+1)/2 \rceil$ honest observers:

$$\Pr[\text{phantom event confirmed}] \leq h \cdot \epsilon_{\mathsf{forge}} \leq n \cdot \epsilon_{\mathsf{forge}} = \mathsf{negl}(\lambda)$$

where the last equality holds because $\epsilon_{\mathsf{forge}} = \mathsf{negl}(\lambda)$ under EUF-CMA security, and n is polynomial in λ.

Thus, the adversary cannot confirm phantom events except with negligible probability.

Relation to Byzantine consensus: This protocol achieves *agreement on event existence* with $n \geq 2f + 1$ parties tolerating f Byzantine faults, matching the optimal bound for Byzantine broadcast [5]. The key difference from consensus is that we do not require agreement on *ordering*—only on set membership. □

Corollary 4.9 (Tight Bound)**.** *The bound $\lceil (n+1)/2 \rceil$ is tight: it is both necessary and sufficient.*

4.3 Trust Hierarchy

We summarize the trust requirements:

Setting	**Trust Requirement**	**Completeness**
Single generator, no observer	—	Impossible
Single TTP	1 honest party	Perfect
n observers, t corrupted	$t < n/2$	Perfect
n observers, $t \geq n/2$ corrupted	—	Impossible

Table 1: Completeness achievability under various trust assumptions.

5 Zero-Knowledge Proofs and Completeness

A natural question is whether zero-knowledge proofs can reduce the trust requirements for completeness. We show that while ZK proofs provide valuable guarantees, they cannot overcome the fundamental completeness barrier.

5.1 Zero-Knowledge Proof Preliminaries

We recall the standard definition of zero-knowledge proofs [12].

Definition 5.1 (Zero-Knowledge Proof System)**.** *A zero-knowledge proof system for an* NP *relation R consists of algorithms* (Setup, Prove, Verify) *satisfying:*

- Completeness*: For all $(x, w) \in R$:* $\Pr[\mathsf{Verify}(pp, x, \mathsf{Prove}(pp, x, w)) = 1] = 1$.
- Soundness*: For all $x \notin \mathcal{L}(R)$ and all* PPT *provers $\mathcal{P}^*$:* $\Pr[\mathsf{Verify}(pp, x, \mathcal{P}^*(pp, x)) = 1] \leq \mathsf{negl}(\lambda)$.
- Zero-knowledge*: There exists a* PPT *simulator $\mathcal{S}$ such that* $\{\mathsf{Prove}(pp, x, w)\} \approx \{\mathcal{S}(pp, x)\}$.

The crucial observation is that a ZK proof can only prove statements about *existing* witnesses. It cannot prove statements about the *non-existence* of additional witnesses unknown to the verifier.

5.2 What ZK Proofs Can Achieve

Definition 5.2 (Verifiable Processing). *An audit system provides* verifiable processing *if the generator can prove that each logged event was processed according to a specified computation, without revealing the computation's internal state.*

Proposition 5.3 (ZK for Verifiable Processing). *Zero-knowledge proofs can achieve verifiable processing: the generator proves "event e_i was processed by algorithm $\mathcal{A}$ and produced output o_i" without revealing intermediate states.*

Proof. Direct application of ZK-SNARKs [12] or ZK-STARKs [13]. Define the NP relation $R_{\mathsf{proc}} = \{((e_i, o_i, \mathcal{A}), w) : w \text{ is a valid execution trace of } \mathcal{A}(e_i) = o_i\}$. The generator constructs a proof π_i such that $\mathsf{Verify}(pp, (e_i, o_i, \mathcal{A}), \pi_i) = 1$ iff $(e_i, o_i, \mathcal{A}) \in \mathcal{L}(R_{\mathsf{proc}})$. □

5.3 What ZK Proofs Cannot Achieve

Theorem 5.4 (ZK Cannot Achieve Completeness). *Zero-knowledge proofs cannot transform a completeness-impossible setting into a completeness-possible one.*

Proof. We prove this by showing that any ZK-based completeness claim reduces to a vacuous statement.

Suppose a protocol Π_{ZK} claims to achieve completeness via ZK proofs. Let R_{comp} be the relation the prover must prove:

$$R_{\mathsf{comp}} = \{(L, w) : w \text{ certifies that } E_{\mathcal{G}}(T) \subseteq L\}$$

For this relation to be well-defined, the witness w must encode information about $E_{\mathcal{G}}(T)$—the set of all generated events.

Case 1: The witness encodes all events. If w contains a representation of $E_{\mathcal{G}}(T)$, then the prover (generator) must know $E_{\mathcal{G}}(T)$. An adversarial generator that omits event e^* simply constructs $w' = E_{\mathcal{G}}(T) \setminus \{e^*\}$ and proves the statement $E_{\mathcal{G}}(T) \setminus \{e^*\} \subseteq L$. This proof is *valid* (the prover is honest about the witness it uses). The ZK soundness property only prevents proving *false* statements—but from the witness's perspective, the statement is true.

Case 2: The witness does not encode all events. If w is not a complete representation, then the relation R_{comp} is undefined—there is no way to verify completeness without knowing what "complete" means.

Information-theoretic argument: The fundamental issue is that $E_{\mathcal{G}}(T)$ (the ground truth) is known only to the generator. The verifier has no independent source of information about $E_{\mathcal{G}}(T)$. ZK proofs are *derivation* mechanisms: they derive statements from existing information. They cannot *create* information about events that were never communicated to any external party.

Formally, let $\mathcal{I}(\cdot)$ denote information content. For any ZK proof π generated by the prover:

$$\mathcal{I}(\pi) \leq \mathcal{I}(pp, x, w)$$

where x is the public input and w is the witness. If $e^* \notin w$ and $e^* \notin x$, then $\mathcal{I}(\pi)$ contains no information about e^*. The verifier cannot detect the omission of e^*. □

Corollary 5.5 (ZK Provides Orthogonal Guarantees). *ZK proofs for audit systems provide:*

- *Correct processing of logged events (✓)*

- *Privacy of internal computations (✓)*
- *Completeness of logging (✗)*

Remark 5.6 (ZK with External Reference Sets). *Our impossibility applies when the generator is the sole source of information about which events exist. If an* external reference set *S is established independently (e.g., a commitment to all inputs published by a TTP before processing begins), then ZK* can *prove completeness* relative to *S: the prover demonstrates that every element of S appears in the log. However, this does not contradict our result—the external reference set S itself requires trust assumptions (a TTP or multi-party protocol) to ensure S is complete. ZK merely shifts the completeness requirement from the log to the reference set; it does not eliminate the need for external trust.*

5.4 Hybrid Approaches

While ZK alone cannot achieve completeness, it can strengthen multi-party protocols:

Proposition 5.7 (ZK-Enhanced Multi-Party Completeness). *In a multi-party setting with $\lceil (n+1)/2 \rceil$ honest observers, ZK proofs can additionally provide:*

1. *Privacy: Observers verify event commitments without seeing event contents.*
2. *Efficient verification: Observers verify batch proofs instead of individual events.*

Proof. Protocol sketch:

1. Generator commits to event: $c_i = \mathsf{Com}(e_i; r_i)$.
2. Generator broadcasts c_i to observers and proves in ZK: "c_i is a valid commitment to an event satisfying policy P".
3. Observers acknowledge commitments (not events).
4. Log contains commitments; events revealed only when needed.

This preserves completeness (observers still verify event existence via commitments) while adding privacy. □

6 Implications for Audit System Design

Our theoretical results have practical implications for the design of audit systems.

Remark 6.1 (Scope of This Work). *This paper establishes* theoretical boundaries*—what is and is not achievable through cryptographic means. We deliberately do not provide implementation-specific guidance (e.g., concrete performance benchmarks, latency analysis for n-observer protocols, or optimal parameter selection). Such engineering considerations depend heavily on deployment context and are orthogonal to the fundamental limits we establish. Our contribution is to clarify* what *trust assumptions are necessary;* how *to implement systems under those assumptions efficiently is a separate engineering challenge.*

6.1 Common Misconceptions

1. **"Hash chains ensure complete logging."**
   False. Hash chains ensure that logged events cannot be modified—they say nothing about events never logged.

2. **"Digital signatures prevent omissions."**
   False. Signatures authenticate what is signed. An unsigned event is simply absent, not detectable as omitted.

3. **"Blockchain provides completeness."**
   Partially true. Blockchain achieves completeness through multi-party consensus among miners/validators. A single participant cannot unilaterally omit transactions because other participants would include them. This is our multi-party model with $t < n/2$ corrupted participants.

4. **"Append-only logs are complete."**
   False. Append-only ensures new entries don't modify old ones. It doesn't ensure all events become entries.

6.2 Design Guidelines

Based on our analysis, we recommend:

1. **Explicitly identify the completeness model.**
   - Who serves as external observer?
   - What are the trust assumptions?
   - What failure modes exist?

2. **Distinguish integrity from completeness claims.**
   - Integrity: achievable with cryptography alone.
   - Completeness: requires external parties.

3. **For applications requiring completeness:**
   - Real-time reporting to a TTP is the simplest solution.
   - Multi-party models require explicit trust relationships.
   - Single-party claims of completeness are unfounded.

4. **Use ZK proofs for processing verification, not completeness.**
   - ZK proves *how* events were processed.
   - It cannot prove *all* events were logged.

6.3 Implementation Trade-offs

We briefly discuss practical considerations for implementing completeness mechanisms.

Completeness vs. Latency. Achieving completeness requires external confirmation before an event is considered "final." In the TTP model, latency equals one round-trip to the trusted party. In the multi-party model with n observers, confirmation requires $\lceil (n+1)/2 \rceil$ acknowledgments, introducing latency proportional to network diameter and observer response time. For high-frequency applications (e.g., algorithmic trading with sub-millisecond requirements), this overhead may be prohibitive.

Completeness vs. Cost. Each event requires $O(n)$ signature verifications in the multi-party model. For $n = 100$ observers and 10^6 events/day, this amounts to 10^8 signature operations daily. Modern Ed25519 implementations achieve $\sim 10^5$ verifications/second on commodity hardware, so a single server can handle this load, but costs scale linearly with event volume and observer count.

Partial Synchrony Considerations. Under partial synchrony (eventual delivery within bound Δ), the protocol must wait Δ time units before declaring an event "not received." If Δ is large (e.g., minutes in geo-distributed systems), confirmation latency increases accordingly. Applications must choose Δ to balance false positives (honest delays misclassified as omissions) against detection delay.

Implementation Checklist. Systems claiming completeness should document:

1. The trust model (TTP, multi-party, or hybrid).
2. The set of observers and their trust assumptions.
3. The confirmation latency and its dependence on network conditions.
4. The failure mode when observers are unavailable or partitioned.
5. The cost model (computation, communication, storage).

7 Extensions and Future Directions

This section discusses extensions to our main results and identifies directions for future research. The first three subsections (Sections 7.1–7.3) present achievable relaxations of completeness. Section 7.4 discusses the open question of economic incentives. Section 7.5 summarizes open problems. Section 7.6 applies our framework to existing systems.

7.1 Relative Completeness with Prior Knowledge

Our impossibility result assumes the verifier has no prior knowledge about what events should exist. If the verifier possesses an independent information source, a weaker form of completeness becomes possible.

Definition 7.1 (Relative Completeness)**.** *Let I be an external information source that provides the verifier with a* reference set $S_I \subseteq E_{\mathcal{G}}(T)$ *of events that should exist. An audit system achieves* completeness relative to I *if:*

$$\Pr\big[\, S_I \not\subseteq L_T \,\big] \leq \mathsf{negl}(\lambda)$$

That is, all events in the reference set appear in the log.

Proposition 7.2 (Relative Completeness is Achievable)**.** *If the verifier has access to an information source I that provides a complete reference set $S_I = E_{\mathcal{G}}(T)$, then relative completeness is achievable via simple set comparison.*

Proof. The verifier checks $S_I \subseteq L_T$ by comparing each element of S_I against L_T. If $S_I = E_{\mathcal{G}}(T)$, this detects all omissions. □

Remark 7.3 (Information Source as Trust Assumption)**.** *The information source I must itself be trustworthy—if I is controlled by the adversary, it provides no guarantee. Examples of information sources:*

- ***External counterparties****: In a trading system, the counterparty to each trade can attest to the transaction.*

- ***Specification documents****: A protocol specification defines expected event types; the verifier can check type completeness (all specified types appear) but not instance completeness (all instances of each type appear).*

- ***Regulatory requirements****: A regulator may specify that certain events must be logged; verifiers can check compliance with these requirements.*

Crucially, relative completeness reduces to our TTP model: the information source I acts as a trusted third party that independently records (or specifies) what events should exist. This is not a workaround to our impossibility result but an instantiation of it—trust is moved from the generator to the information source.

7.2 Probabilistic Completeness

7.3 Probabilistic Completeness

While perfect completeness is impossible without trust, we can achieve *probabilistic* guarantees through sampling.

Definition 7.4 (Spot-Check Completeness)**.** *An audit system achieves (p, ϵ)-spot-check completeness if:*

- *An external auditor samples events with probability p.*
- *Any omission is detected with probability $\geq 1 - \epsilon^{1/p}$.*

Proposition 7.5. *Spot-check completeness is achievable and reduces auditor burden while providing probabilistic guarantees.*

Proof. If auditor samples with probability p and generator omits k events:

$$\Pr[\text{at least one omission detected}] = 1 - (1-p)^k \geq 1 - e^{-pk}$$

For k large, detection is near-certain even with small p. □

This suggests a practical trade-off: perfect completeness requires continuous observation, but high-probability detection requires only sampling.

7.4 Economic Incentives

Can economic mechanisms substitute for trust?

Conjecture 7.6 (Incentive-Compatible Completeness)**.** *Under appropriate economic incentives (bonds, penalties, reputation systems), rational (non-Byzantine) generators may achieve completeness without trusted observers.*

This conjecture is challenging for several reasons:

- **Undetectable omissions**: If omissions are undetectable (our main theorem), how can penalties be imposed? Incentive mechanisms typically require observable violations.

- **Benefit-cost asymmetry**: The benefit of omitting incriminating evidence may exceed any bond or penalty, especially in high-stakes scenarios (e.g., concealing fraud).

- **Sybil attacks**: Reputation systems can be gamed by creating multiple identities.

- **Collusion**: In multi-party settings, colluding parties can share the economic surplus from successful omissions.

Blockchain systems partially address this via mining rewards and slashing conditions, but these mechanisms assume a competitive market of block producers—a form of multi-party trust. Whether purely economic mechanisms can *substitute* for (rather than complement) trust assumptions remains open. This connects to mechanism design and cryptoeconomics, which are outside our scope.

7.5 Open Problems

1. **Optimal sampling strategies**: What sampling probability minimizes auditor cost while achieving (p, ϵ)-completeness?
2. **Anonymous observers**: Can completeness be achieved with privacy-preserving observation (observers don't learn event contents)?
3. **Asynchronous completeness**: How do network delays affect multi-party completeness guarantees?
4. **Rational vs. Byzantine**: What completeness guarantees are possible against rational (economically motivated) vs. Byzantine (arbitrarily malicious) adversaries? A promising direction is integration with mechanism design: if omission detection triggers economic penalties (bond slashing, reputation loss), rational generators may be incentivized toward completeness even without trusted observers. This would require formalizing "detectable omission" in a game-theoretic framework.
5. **Multiple independent generators**: Our model considers a single generator. In distributed systems, multiple independent generators may produce events. Does completeness for individual generators compose? Specifically, if generators $\mathcal{G}_1, \ldots, \mathcal{G}_m$ each achieve completeness independently, does the combined system achieve completeness? We conjecture that composability holds if generators are truly independent, but cross-generator omissions (where $\mathcal{G}_i$ omits an event because it colludes with $\mathcal{G}_j$) require joint observation.

7.6 Case Studies

We briefly analyze how our framework applies to existing systems.

Certificate Transparency (RFC 6962). CT [2] uses append-only Merkle trees to log TLS certificates. In our terminology:

- *Generator*: Certificate Authorities (CAs) issuing certificates.
- *Observers*: Public CT log servers that receive and store certificates.
- *Verifiers*: Monitors and auditors that check logs for completeness.

CT achieves completeness through a multi-party model: CAs must submit certificates to multiple independent logs, and browser policies enforce that certificates not appearing in logs are rejected. The trust assumption is that *at least one* log server is honest and that monitors actively watch for omissions. This matches our TTP model (Section 4) where the log server acts as the trusted party. CT's "Maximum Merge Delay" (MMD) corresponds to our Δ parameter in partial synchrony.

Financial Audit Logs (SEC Rule 17a-4). SEC Rule 17a-4 requires broker-dealers to maintain records in non-rewritable, non-erasable format ("WORM" storage). This achieves *integrity* in our terminology—logged records cannot be modified. However, it does *not* achieve completeness: a malicious broker can simply not log certain transactions. Our framework explains why WORM storage alone is insufficient and why regulators additionally require real-time trade reporting to central authorities (FINRA's OATS, CAT)—these authorities serve as TTPs that independently record events.

Blockchain Systems (Bitcoin, Ethereum). Blockchain achieves completeness for *broadcast transactions* through multi-party consensus. In our model:

- *Observable events*: Transactions broadcast to the mempool.
- *Internal events*: Transactions a user generates but does not broadcast.

Miners/validators serve as observers; the $> 50\%$ honest mining power assumption corresponds to our honest majority requirement. Notably, blockchain cannot guarantee completeness for transactions users choose not to broadcast—this is precisely our impossibility result. Mempool censorship (miners refusing to include valid transactions) is an observable-event attack addressed by proposer-builder separation and related mechanisms.

8 Conclusion

We have established fundamental limits on what cryptography can achieve for audit log completeness. Our main result—that completeness is impossible without external trust—is not a limitation of current techniques but an information-theoretic barrier. We characterized the minimal trust required: a single TTP or honest majority among multiple observers. We showed that zero-knowledge proofs, while valuable for processing verification, cannot overcome the completeness barrier.

These results clarify the boundaries of cryptographic guarantees. Claims of "cryptographic completeness" without identifying the trust model should be viewed skeptically. Conversely, systems that explicitly incorporate trusted observers are on solid theoretical footing.

We hope this work provides a foundation for rigorous analysis of audit system security properties and guides the design of future systems that clearly delineate their trust assumptions.

Acknowledgments

The author thanks the anonymous reviewers for their valuable feedback on earlier drafts of this work.

References

[1] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378. Springer, 1987.

[2] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[4] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[5] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[6] S. A. Crosby and D. S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security Symposium*, pages 317–334, 2009.

[7] D. Ma and G. Tsudik. A new approach to secure logging. *ACM Transactions on Storage*, 5(1):2:1–2:21, 2009.

[8] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[9] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[10] R. Küsters, T. Truderung, and A. Vogt. Accountability: Definition and relationship to verifiability. In *ACM CCS*, pages 526–535, 2010.

[11] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *PODC*, pages 27–30, 1983.

[12] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326. Springer, 2016.

[13] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR ePrint*, 2018.

[14] G. Caldarelli. Understanding the blockchain oracle problem: A call for action. *Information*, 11(11):509, 2020.

A Proof Details

A.1 Formal Statement of Theorem 3.1

Theorem (Restatement)**.** *Let* $\Pi = (\mathsf{Setup}, \mathsf{Log}, \mathsf{Verify})$ *be an audit system. Let* $\mathcal{G}$ *be an event generator controlled by an omission adversary* $\mathcal{A}_{\mathsf{omit}}$*. Let* View_V *denote the verifier's view.*

For any verification strategy V*:*

$$\Pr[V \textit{ detects omission} \mid \mathcal{A}_{\mathsf{omit}} \textit{ omits event}] = \Pr[V \textit{ detects omission} \mid \textit{no omission}]$$

That is, the verifier's detection probability is independent of whether an omission occurred.

Proof. Let $E = \{e_1, \ldots, e_n\}$ be the events generated, and $S \subseteq E$ be the omitted set.

In World 1 (no omission): $\mathcal{G}$ generates $E \setminus S$ and logs all of them. In World 2 (omission): $\mathcal{G}$ generates E and logs only $E \setminus S$.

The log L is identical in both worlds. The verifier's view $\mathsf{View}_V = (pp, L)$ is identical in both worlds.

By definition of computational indistinguishability (here, perfect indistinguishability):

$$\mathsf{View}_V^{\text{World 1}} \equiv \mathsf{View}_V^{\text{World 2}}$$

Therefore, any deterministic or probabilistic strategy V produces identical output distributions in both worlds, and detection probability is independent of omission. □

A.2 Proof of Theorem 4.7 (Tight Bound)

We provide a more detailed proof of the multi-party lower bound.

Proof. Let there be n observers and $h = \lceil (n+1)/2 \rceil - 1$ honest observers. Then the number of corrupted observers is $c = n - h = n - \lceil (n+1)/2 \rceil + 1 \geq \lfloor n/2 \rfloor + 1 > n/2$.

Adversary strategy:

1. $\mathcal{A}$ controls the generator and c observers.
2. $\mathcal{A}$ generates event e^* and sends it to all observers.
3. Honest observers record e^* and sign acknowledgment.
4. $\mathcal{A}$ instructs corrupted observers to:
    - Delete e^* from their records.
    - Sign a statement: "I did not receive e^*".
5. $\mathcal{A}$ omits e^* from the generator's log.

Verifier's dilemma:

- $h < n/2$ observers claim e^* exists.
- $c + 1 > n/2$ parties (corrupted observers + generator) claim e^* doesn't exist.

Without additional information, a majority vote concludes e^* doesn't exist. The verifier cannot identify which observers are honest a priori. Therefore, completeness fails with non-negligible probability. □