Cryptographic Attestation Against Integrity Attacks in Service Monitoring: A Threat Model and Verifiable Architecture

M A Anusuya[1], Chayadevi M L[2], Shraddha C[3], Vani H Y[4]*
Department of Computer Science and Engineering,
JSS Science and Technology University (JSS STU), Mysuru, Karnataka, India[1]
B N M Institute of Technology, Visvesvaraya Technological University, Bengaluru, Karnataka, India[2]
Vidyavardhaka College of Engineering, Visvesvaraya Technological University, Mysuru, Karnataka, India[3]
Department of Information Science and Engineering,
JSS Science and Technology University (JSS STU), Mysuru, Karnataka, India[4]

***Abstract*—Service uptime monitoring infrastructure is a high-value target for data-integrity attacks: a single compromised or dishonest monitoring provider can fabricate availability records, retroactively suppress outage evidence, or silently alter historical data, and clients today have no cryptographic means of detecting such manipulation. This study develops a threat model for monitoring-data integrity attacks—covering provider-side tampering, evidence suppression, Sybil-identity flooding, and submission replay—and presents a verification architecture engineered to resist each threat in that model. Independent validator nodes sign availability observations with Ed25519 keys; a quorum-based aggregation rule tolerates up to f < Q/2 Byzantine validators, SHA-256 content-hash commitments bind off-chain evidence to an immutable on-ledger record that any third party can independently re-derive and check without trusting the aggregator; and stake-bonded registration imposes a quantifiable capital cost on Sybil identities. We formalize the adversary model, prove signature unforgeability under the Elliptic Curve Discrete Logarithm assumption, derive the capital cost of quorum capture, and bound the residual attack surface—selective evidence inclusion and round-stalling—that persists even under a semi-honest aggregator. A seven-day, five-validator, three-region deployment achieves 99.7% quorum agreement, sub-6-second worst-case attestation latency, and zero false positives or negatives across 200 independently re-verified historical rounds, confirming that the architecture removes the central point of trust that lets a single compromised provider corrupt monitoring evidence undetected.**

Keywords*—Cryptographic attestation; threat modelling; Byzantine fault tolerance; Sybil resistance; digital signatures; data integrity; tamper-evident logging; adversarial security analysis; non-repudiation

I. Introduction

Modern businesses depend on the continuous availability of their digital services, and Service Level Agreements (SLAs)[1] translate that dependence into a measurable contract, with an understanding that a 99.9% SLA permits under nine hours of downtime per year, while a 99.99% SLA shrinks that allowance to roughly fifty-three minutes. The party measuring compliance against that contract, however, is almost always either the service operator itself or a third-party monitoring vendor such as Pingdom or Uptime Robot.

In both cases, the measuring party has a direct interest in the outcome it reports. This is a textbook conflict-of-interest trust problem stating the "entity being audited also controls the audit evidence".

Architecturally, this trust problem reduces to a single point of compromise. A monitoring vendor's backend database is the sole record of historical availability. If that database is breached, manipulated by an insider, or instructed by the vendor to selectively omit incidents would trigger SLA penalties. Hence, no independent party has any way to detect the tampering after the fact. Re-running the same HTTP checks later does not help, because the original observation is the only artifact that could prove or disprove what actually happened at the time. It was never cryptographically bound to anything. So, the result is an evidentiary integrity gap stating that uptime claims are unfalsifiable assertions rather than independently verifiable facts.

This is fundamentally a non-repudiation and tamper evidence problem, and it admits a direct cryptographic treatment. If observations are digitally signed by the party that made them, an aggregator can no longer alter a validator's claim without invalidating the signature, if the aggregated report is committed via a collision-resistant hash to a public, append-only, Byzantine-fault-tolerant ledger, no party, including the aggregator itself, can quietly rewrite history after the commitment is published.

This study treats such a ledger as an interchangeable security primitive rather than as the contribution in itself. The proposed approach instantiates it using Solana[2] for its sub-second finality and negligible per-transaction cost and instantiates evidence storage using the IPFS content-addressing layer, but the security properties analysed in Section VI hold for any primitive offering the same finality and collision-resistance guarantees.

The contributions of this study are: A formal adversary model and threat taxonomy for monitoring data integrity attacks are presented in Section I. Section II discusses a

*Corresponding author.

cryptographic verification architecture in terms of independent signed observation, quorum-based Byzantine-fault-tolerant aggregation, hash-committed ledger anchoring, and stake-bonded Sybil resistance engineered against adversary effects. Section III reviews related security and attestation literature applied in the cybersecurity field. Formal derivations of the resulting security guarantees, including signature unforgeability, the capital cost of quorum capture, and the residual attack surface under a semi-honest aggregator are discussed in Section IV and V. Section VI and VII discusses abound an empirical evaluation of attestation latency, cost, quorum agreement, and independent verification performance under a real seven-day, multi-region deployment Section VIII compares the resulting security posture against centralized alternatives, and lastly conclusion and future enhancements are presented in Section IX.

Margins, column widths, line spacing, and type styles are built-in; examples of the type styles are provided throughout this document and are identified in italic type, within parentheses, following the example. Some components, such as multi-level equations, graphics, and tables, are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow.

Objectives:

- To design a decentralized, cryptographically verifiable service uptime monitoring framework that eliminates dependence on a single trusted monitoring provider and enables transparent, tamper-evident verification of monitoring records.
- To ensure the integrity and authenticity of uptime monitoring data by employing independent validator nodes with Ed25519 digital signatures, quorum-based Byzantine fault-tolerant consensus, and SHA-256 hash commitments for immutable evidence verification.
- To evaluate and validate the security and performance of the proposed framework through formal security analysis and experimental deployment, demonstrating resistance to data tampering, Sybil attacks, evidence suppression, and replay attacks while maintaining high monitoring accuracy and low attestation latency.

II. Adversary Model and Threat Taxonomy

A. System Roles and Trust Assumptions

We define the system as a tuple $\Pi = \langle V, H, C, S, A \rangle$, where V is the set of independently operated validator nodes, H is the aggregator (hub), C is the on-ledger commitment program, S is the content-addressed evidence store, and A is the client-facing verification interface. Critically, H is not assumed to be honest. We adopt a semi-honest aggregator model: H is assumed to follow the wire protocol—it cannot forge a validator's signature or fabricate ledger state — but it is not trusted to include every valid submission impartially, to finalize rounds promptly, or to act in the client's interest. This is a strictly weaker trust assumption than the one placed on centralized monitoring vendors today, where the equivalent of H is fully trusted with no cryptographic backstop at all.

B. Adversary Capabilities and Non-Capabilities

We consider a probabilistic polynomial-time adversary with the following capabilities: 1) X may corrupt up to f validators where $f < Qo/2$, with Qo the quorum threshold (a Byzantine-minority assumption) 2) X may fully control the network path between validators and H, including dropping, delaying, reordering, or duplicating messages; 3) X may operate H itself under the semi-honest model defined above 4) X may observe all public ledger state and all content-store data, since both are public by design. X is assumed unable to: 5) forge an Ed25519 signature without the corresponding private key, under the Elliptic Curve Discrete Logarithm (ECDLP) hardness assumption; 6) find a SHA-256 collision or pre-image; or 7) violate the liveness or finality guarantees of the underlying ledger's own consensus mechanism, which is treated here as an external, already-analysed primitive rather than re-derived from first principles.

C. Threat Taxonomy

Standard systems-security-engineering practices [3][4][5] are derived directly from the adversary capabilities of Section II-B rather than from an ad hoc list of concerns as shown below. Table I structures the remainder of the study: Sections IV–V describe the mechanisms named in the Défense column, and Section VI proves that each mechanism achieves the property assigned to it here.

TABLE I. Threat Taxonomy

ID	Threat (Attacker)	Defense (§)
T1	Signature forgery/identity spoofing (network or external attacker)	§VI-A
T2	Sybil-identity flooding to bias the quorum (capital-bounded adversary)	§VI-B
T3	Historical evidence tampering(aggregator, insider, or breach)	§VI-C, §VI-D
T4	Selective inclusion/evidence suppression (semi-honest H)	§VI-E
T5	Submission replay across rounds (network attacker)	§VI-F
T6	Round-stalling / availability denial (semi-honest H)	§VI-E

III. Related Work

A. Trust Assumptions in Centralized Monitoring Services

Centralized uptime monitoring vendors like Pingdom [6], Uptime Robot [7], and Datadog Synthetics [8] are the most widely deployed web-based Software-as-a- Service (SaaS) monitoring platforms. They run availability checks from their own infrastructure and expose results through dashboards and APIs backed by private databases. From a security standpoint, this places the entire evidentiary record inside a single trust and administrative boundary. The same vendor is responsible for performing the compliance check, storing the results, and generating the compliance report. Because all of these activities are handled by a single party, there is no independent mechanism to verify or audit the complete chain of custody

from the initial observation to the final reported outcome. The threat model in Section II directly targets this boundary. The architecture in Sections IV – V gives clients a cryptographic chain of custody that does not depend on trusting the party that produced it.

B. Tamper-Evident Attestation and Public Ledgers

Public, Byzantine-fault-tolerant ledgers have been studied extensively as tamper-evident attestation primitives. Androulaki et al. [9] describe Hyperledger Fabric's permissioned model, in which all participants are known and individually accountable. This is an assumption that does not hold in an open monitoring setting where validators are anonymous, self-registering parties. Buterin [10] introduced smart-contract-capable public ledgers with Ethereum, establishing on-chain program execution and data anchoring as a general security primitive, albeit at limited throughput. This work instantiates the ledger primitive using Solana [11][12], whose Proof-of-History-based consensus offers sub-3-second transaction finality — a security-relevant property here insofar as it bounds the window during which a finalized observation could be contested or double-anchored. Its throughput characteristics are treated as an implementation detail rather than a security contribution. Benet [13] establishes the IPFS content-addressing scheme, used here as a complementary tamper-evidence mechanism. According to this mechanism, any modification is stored, and it acts as evidence by changing its content address, making silent substitution self-detecting (Section VI-D).

C. Stake-Bonded Sybil Resistance and Oracle Security

Decentralized oracle networks such as Chainlink [14] establish the pattern of requiring node operators to stake collateral forfeitable under provable misbehavior, giving a dishonest report a quantifiable economic cost rather than relying on reputation alone. Our work adapts the pattern to HTTP availability attestation, where the adversarial goal is not price manipulation, but quorum capture used in registering enough Sybil identities to bias the majority vote on a binary up signal. Section VI-B derives the capital cost of quorum capture under our staking parameters, extending the Sybil-resistance line of work that traces back to the Byzantine Generals' Problem [15] to this specific attack.

IV. Materials and Methods

Fig. 1 illustrates the four architectural layers and the trust boundaries between them. Every arrow that crosses a boundary in Fig. 1 corresponds to at least one entry in the threat taxonomy of Table I; the hub and queue layer shown in the figure are treated as a single logical aggregator H in the text that follows, consistent with the system definition in Section II-A.

All cross-boundary communications in the proposed architecture include validator-to-hub, hub-to-ledger, and hub-to-storage message exchanges evaluated against the threat taxonomy presented in Table I.

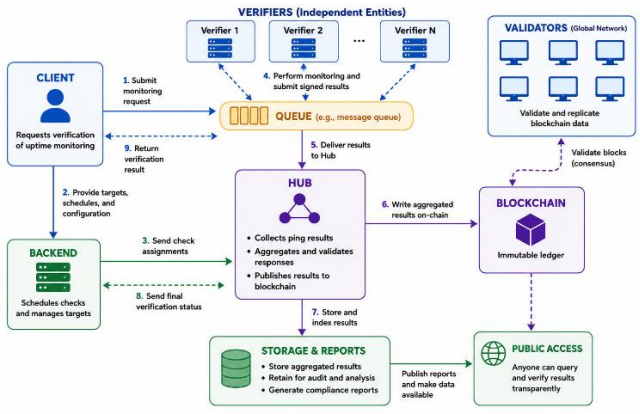

Fig. 1. System security architecture.

A. Validator Node Architecture

Each validator $v \in V$ is an independently operated process, run by a self-selected, anonymous operator on commodity hardware — exactly the class of party the threat model of Section II-B treats as potentially Byzantine. Each validator maintains a persistent Ed25519 key pair (sk_v, pk_v) that is the sole basis for its accountability: every claim it makes is non-repudiably bound to pk_v. The validator state machine cycles through four phases per round: IDLE → PULLING → CHECKING → SUBMITTING.

During PULLING, the validator authenticates with the aggregator by signing a challenge σ_{auth}=Sign(sk_v, pk_v// timestamp) [16], and submits (pk_v, timestamp, σ_{auth}) to the task-pull endpoint. The aggregator verifies the signature using pk_v, a task payload containing the target URL and round timestamp is returned only if the validator is registered, active, and the signature is valid.

During CHECKING, the validator sends an HTTP HEAD request to the target URL with timeout τ (default 10s). The result is a tuple (is_up, latency_ms), where is_up indicates whether a 2xx response was received within τ. Algorithm 1 specifies the full signed submission procedure.

Algorithm 1: Signed Observation Submission

Input:sk_v,pk_v,target_url,round_timestamp,timeout τ
Output: Signed submission payload

t_start ← current_time()
response ← HTTP_HEAD(target_url, τ)
latency ← current_time() − t_start
if response.status ∈ [200, 299] then
 is_up ← 1
else
 is_up ← 0
end if
payload ← {target_id:SHA256(normalize(target_url)), round_ts: round_timestamp,is_up: is_up, latency: latency, pubkey: pk_v }
σ ← Sign(sk_v, serialize(payload))

B. Aggregator (Hub) Architecture

The aggregator H is the architecture's semi-honest party (Section II-A) and the closest analogue to a traditional monitoring vendor's backend. The difference is that its outputs are independently checkable rather than trusted outright. H maintains three internal data structures: a round-state table mapping (target_url, round_timestamp) to received submissions, an expected_validators registry of active validator public keys; and a round scheduler keyed on target_id and minimum subscription interval.

Upon receiving a validator submission (payload, σ), the aggregator executes the verification pipeline in Algorithm 2. A submission is accepted if and only if all four checks succeed; partial passes are rejected, which is the mechanism that closes threat T1 at the aggregator boundary and threat T5 in combination with the round scoping described in Section V-A3.

Algorithm 2: Submission Authentication and Validation Pipeline

```
Input: submission S = (payload, σ)
Output: ACCEPT | REJECT

pk_v ← payload.pubkey
// Check 1: Signature validity
if ¬Verify(pk_v, serialize(payload), σ)
   return REJECT
round ←rounds[payload.target_id+payload.round_ts]
// Check 2: Validator is registered
if pk_v ∉ round.expected_validators
   return REJECT
// Check 3: No duplicate for this round
if pk_v ∈ round.submissions
   return REJECT
// Check 4: Round is still open
if round.state = CLOSED
   return REJECT
round.submissions.add(S)
return ACCEPT
```

Region-wise aggregates are computed by mapping the source IP of each submission to a continent via a GeoIP database; region is stored in the evidence report but not anchored on-chain, to minimize chain state. The aggregator triggers round finalization when either the count of accepted submissions reaches the quorum threshold Q (default 3) or the round timeout T_round elapses (default 120 s). Algorithm 3 specifies finalization: building the JSON evidence report, committing it to the content-addressed store, anchoring the commitment on-ledger, and distributing validator rewards.

Algorithm 3: Quorum Finalization and Commitment Procedure

```
Input: round_id, submissions S = {s_1,...,s_n}
Output: content CID, ledger tx_hash

uptime ← (Σ s_i.is_up) / |S| × 100
latencies ← sort([s_i.latency for s_i ∈ S])
median_lat ← latencies[⌊ |S|/2 ⌋ ]
report ← build_json(S, uptime, median_lat)

cid ← contentstore_upload(report)
report_hash ← cid

tx ← build_ledger_tx(instruction:submit_round,
target_id:round.target_id,round_ts: round.timestamp,
uptime_pct:uptime,median_lat:median_lat,report_hash: report_hash)

tx_hash ← ledger_send(tx)
distribute_rewards(S)
```

C. Tamper-Evident Commitment Layer (Ledger Anchoring)

The commitment layer is the architecture's tamper-evident backstop: once a round is anchored, altering it requires breaking the security of the underlying ledger itself, not merely compromising the aggregator. The Solana program instantiating this layer manages five account types: a Target Account (PDA seeded by ["target", target_id]) storing the normalized URL hash, a ValidatorAccount (PDA seeded by ["validator", pk_v]) holding stake and active status; a StakePool (PDA seeded by ["stake_pool"]) acting as the escrow holding staked SOL; a RoundSummaryAccount (PDA seeded by ["round", target_id, round_timestamp]) storing the finalized round commitment; and a RewardVault (PDA seeded by ["reward_vault"]) holding funds for validator rewards.

The Round Summary Account stores: target_id [32 bytes], round_timestamp [8 bytes], uptime_percent [4 bytes, fixed-point ×100], median_latency_ms [4 bytes], report_hash [32 bytes, SHA-256] — 80 bytes total, requiring a rent-exempt deposit of approximately 0.0015 SOL. PDA derivation follows Solana's canonical procedure: given program ID P and seeds ["round", target_id, round_ts], the address A is derived as in Eq. (1).

$$A=\text{find\_program\_address}([\text{“round”}//\text{target\_id}//\text{round\_ts}],P) \quad (1)$$

where find_program_address iterates a nonce from 255 downward until SHA-256(seeds//nonce//P) maps to a point not on the Ed25519 curve. This guarantees A cannot itself be a valid signing key, so the commitment address can only ever be controlled by the program — a security property of the derivation, not merely an addressing convenience, since it rules out an adversary pre-computing or squatting a colliding commitment address.

D. Content-Addressed Evidence Store

Each round's evidence report is stored as a deterministically serialized JSON document. Every approved validator appears once in the submissions list, with a base58-encoded validator_pubkey, the measured latency_ms, a Boolean is_up flag, and a cryptographic signature covering the payload. The aggregation section records uptime_percent (fixed-point, four-decimal precision) and median_latency_ms. Content integrity is guaranteed by IPFS's CIDv1 addressing scheme [7], in which the address is derived directly from the content via Eq. (2).

$$CID=\text{Multibase}(\text{Multicodec}(\text{Multihash}(\text{SHA2-256},\text{content}))) \quad (2)$$

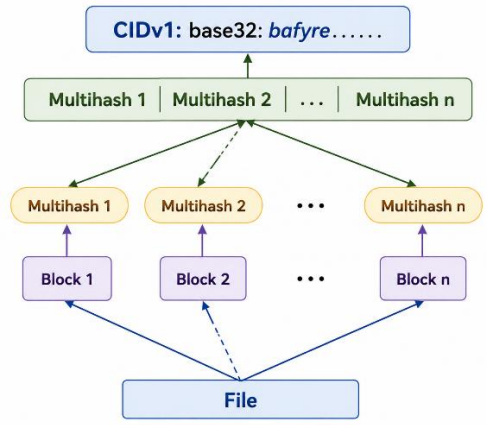

Fig. 2. Content addressing.

Fig. 2 illustrates how a file is split into n blocks, each hashed into a multihash, which are then combined into a root multihash to produce the CIDv1. The key security property this provides is that any modification of the stored evidence — by the aggregator, an IPFS gateway, or any other party — produces a different address, so silent substitution is self-detecting rather than requiring an external auditor to notice.

The above figure shows the file blocks are hashed into multi-hashes and combined into a CIDv1, so any modification to the stored evidence changes its address. An independent verifier checks that the fetched evidence document matches the on-ledger commitment by comparing report_hash on-ledger against the recomputed hash of the fetched document (Section V-C, Algorithm 4), eliminating the need to trust the content store gateway.

V. Protocol Methodology

This section specifies the protocol mechanics from the three perspectives introduced in Section II-A: the aggregator, which runs round scheduling, validator authentication, and on-ledger commitment; the validator, which performs availability measurements and submits signed observations; and the client, which independently verifies the resulting evidence without trusting the aggregator.

A. Aggregator-Side Lifecycle

A full observation cycle moves through five stages: SCHEDULED → OPEN → COLLECTING →FINALIZING → ANCHORED.

1) Target registration and URL normalization: Two URLs that point to the same resource must normalize to an identical target_id. Normalization applies, in fixed order: 1) lower-casing the scheme and host; 2) removing default ports (80/443) for HTTP/HTTPS; 3) removing trailing slashes from the path. The identifier is then computed as in Eq. (3).

$$\text{target\_id} = \text{SHA-256}(N(\text{url})) \tag{3}$$

This 32-byte identifier seeds every PDA related to the target, partitioning the on-ledger address space by URL with no collision risk under SHA-256 pre-image resistance — the same property that, in Section VI-D, prevents an adversary from binding a forged report to an existing commitment.

2) Subscription model and round scheduling: Users subscribe at discrete intervals (5, 10, 15, or 60 minutes). Let Sub(t) denote active subscriptions for target *t*, the effective monitoring interval is given by Eq. (4).

$$I_{eff}(t) = \min\{ I_s : s \in \text{Sub}(t) \} \tag{4}$$

A round is scheduled for target *t* at time *T* iff $T \bmod I_{eff}(t) = 0$. A subscriber with a 30-minute interval observes one in every six 5-minute buckets, filtered client-side by $(T - T_{start}) \bmod I_s = 0$, avoiding server-side per-subscription round generation.

3) Task pull authentication: A validator constructs M_{auth} = pk_v // T (T = current Unix timestamp) and submits (pk_v, T, σauth). The aggregator accepts iff: Verify(pk_v, M_{auth}, σ_{auth}) = true, $|T_{hub} - T| \leqslant \Delta_{auth} = 120$ s (clock-skew tolerance), pk_v ∈ expected_validators, and pk_v has not already pulled a task for the current round. This per-round pull restriction — combined with the timestamp window — is the basis of the replay-prevention guarantee proved in Section VI-F: an authentication message carries no round identifier, so its validity window is bounded purely by the clock-skew tolerance, and any reuse outside that window is rejected at Algorithm 2, step 13.

4) Quorum computation and uptime aggregation: Let $Sr = \{s_1, \ldots, s_n\}$ be the accepted submissions for round r, $n \geq Q$. Uptime U*r* is computed by Eq. (5) and median latency L*r* by Eq (6).

$$Ur = (\Sigma s_i.\text{is\_up} / n) \times 100 \tag{5}$$

$$Lr = \text{sort}(\text{latencies})[\lfloor n/2 \rfloor] \tag{6}$$

These values are stored on-ledger as fixed-point u32 integers — an uptime of 99.73% is stored as 9973, avoiding floating-point representation issues, since the Solana runtime supports only integer arithmetic.

5) Evidence report construction and hash commitment: The evidence report is constructed with a fixed field order (target_id, round_timestamp, submissions, aggregation), with submissions sorted lexicographically by validator_pubkey, so any two parties hashing the same logical report obtain identical digests. The on-ledger commitment is computed by Eq. (7).

$$Hr = \text{SHA256}(\text{canonical\_serialize}(\text{report}r)) \tag{7}$$

The CID and H*r* serve as complementary commitments: the CID authenticates content within the content-store network, while H*r* anchors the same commitment immutably on-ledger — the binding proved formally in Section VI-D.

6) Account life cycle and storage reclamation: Once 24 hours have elapsed past round_timestamp, the aggregator runs close_round: the reserved rent (≈0.0015 SOL) is returned to a pre-set reclaim address and the PDA is closed. The full report remains retrievable from the content store, with the closed account's prior on-ledger activity still visible in the ledger's transaction history, minimizing on-ledger storage while

keeping historical evidence reachable and the commitment auditable.

B. Validator-Side Observation Procedure

Each validator checks reachability by sending an HTTP HEAD request and recording the round-trip time as latency. The site is counted as up only if it answers within timeout τ (default 10 s) with a 2xx status; a late reply, a non-2xx status, a redirect, or any local error all set is_up to zero. This deliberate strictness — treating ambiguous outcomes as down rather than up — prevents an under-reporting bias that would otherwise favor whichever party controls the validator.

C. Independent Client-Side Verification

Unlike centralized services, any client can independently re-verify a historical round using only the target URL, the round timestamp, the commitment program ID, and the publicly retrievable evidence report — with no server-side cooperation required beyond serving that already-public data. Algorithm 4 specifies the procedure.

Algorithm 4: Independent Cryptographic Verification

```
Input:target_url,round_timestamp,program_id, evidence_report
Output: VALID | INVALID | MISSING

target_id ←SHA256(normalize(target_url))
    pda←find_program_address(["round",target_id,
round_ts],program_id)
  account ← Ledger_RPC.getAccount(pda)
if account = null then return MISSING
  end
  summary ← deserialize(account.data)
  computed_hash←SHA256(canonical_serialize( evidence_report))

if computed_hash ≠ summary.report_hash
  then
return INVALID // hash mismatch
end if
for each submission s in
   report.submissions do
   payload ← build_payload(s)
   if ¬Verify(s.pubkey, payload, s.sig)
   return INVALID // bad signature
   end if
end for
n ← |report.submissions|
computed_uptime ← (Σ s.is_up / n) × 100
lats ← sort([s.latency for s in submissions])
computed_median ← lats[⌊ n/2⌋ ]
if |computed_uptime − summary.uptime| >ε
 then
return INVALID // aggregation mismatch
end if
if computed_median ≠ summary.median_latency
return INVALID
end if
return VALID
```

Step 21 tolerates a small ε because the on-ledger value is stored as ⌊ U_r × 100⌋ / 100, so rounding error never exceeds 0.01%. Steps 11–16 verify exactly n Ed25519 signatures for n submissions, giving the verifier an O(n) procedure that requires no trust in the aggregator at any step — each check independently rederives a value the aggregator claimed, rather than asking the aggregator to vouch for it.

VI. Security Analysis and Formal Guarantees

This section proves that each defense mechanism introduced in Sections IV–V achieves the security property assigned to it in the threat taxonomy of Table I, under the adversary model of Section II-B.

A. Signature Unforgeability (T1)

Under the ECDLP assumption, no probabilistic polynomial-time adversary can produce a valid submission (payload, σ) for validator identity pk_v without knowledge of sk_v, except with negligible probability. Ed25519 satisfies Existential Unforgeability under Chosen-Message Attack (EUFCMA). A forged signature requires computing σ =Sign(sk_v, m) for some message m without sk_v, which is equivalent to solving ECDLP on the Ed25519 curve — computationally infeasible under the stated assumption. If ECDLP remains hard, any forgery attempt succeeds only with probability negl(λ), λ = 128 being the security parameter. This directly closes T1: an adversary controlling the network (capability 2) in Section II-B) can intercept and replay messages but cannot fabricate a new signed claim attributed to a validator it has not corrupted.

B. Sybil Resistance (T2)

An adversary aiming to bias a round's aggregate toward a target uptime U_target needs sufficiently many of the Q quorum slots reporting falsely; if honest participants report accurately, the adversary requires at least ⌈ Q × f⌉ of Q slots, where f is the vote fraction needed to shift the aggregate past U_target. To hold a strict majority of the quorum, the adversary must register and stake at least C_sybil SOL, given by Eq. (8).

$$C_{sybil} = \lceil (Q + 1) / 2 \rceil \times S_{min} \quad (8)$$

With Q = 5 and S_min = 0.5 SOL, the cost of quorum capture is 1.5 SOL; doubling Q to 10 raises it to 3 SOL. Cost scales linearly with Q, giving an operator a tunable security parameter. Critically, the staked capital must remain locked for the duration of an attack, so a sustained manipulation campaign carries an ongoing capital-opportunity cost rather than a one-time fee — this is what distinguishes T2's mitigation from a static identity check, and is the property that generalizes the Sybil-resistance argument from [8] and [11] to a quorum-vote setting rather than a price-feed setting.

C. Commitment Immutability (T3a)

Once data is anchored in a Round Summary Account, only the program that owns the account can mutate it, and that program's upgrade authority is revoked (set to none) at deployment, foreclosing future code changes that could alter write semantics. Anchor's init constraint [10] further guarantees that a given PDA address can be initialized exactly once, so an adversary cannot re-run initialization with identical seeds to overwrite a 7 existing commitment. Together, these close the part of T3 that targets already-anchored evidence: an

adversary with capabilities 1–4 from Section II-B, but without an upgrade-authority key that no longer exists, cannot alter a finalized round.

D. Report Hash Binding (T3b)

The hash H_r anchored on-ledger refers to exactly one evidence document, because SHA-256 collision resistance means two different reports producing the same H_r would require breaking that assumption — explicitly excluded by adversary non-capability 6 in Section II-B. Substituting the off-ledger evidence document after anchoring is therefore self-defeating: at step 8–9 of Algorithm 4, the recomputed hash of the substituted document will not match summary.report_hash, and verification returns INVALID. This is the mechanism that closes the off-ledger half of T3: even though the evidence document itself lives outside the ledger, in a content store an adversary might control or compromise, the on-ledger commitment makes any substitution immediately and independently detectable, rather than requiring the verifier to trust the store.

E. Aggregator Manipulation Bounds: Selective Inclusion and Round-Stalling (T4, T6)

Even a dishonest aggregator operating under the semi-honest model of Section II-A is bounded in what it can achieve. It cannot forge a validator signature (§VIA), it cannot alter an already-anchored PDA (§VI-C), and it cannot substitute the evidence document without detection (§VI-D). Two attack vectors remain open, and this study deliberately bounds rather than eliminates them: (T4) selective inclusion of validator submissions, and (T6) stalling a round's progress. For T4, the architecture provides only a detective control, not a preventive one: because validator registries are public, a pattern of repeated, systematic exclusion of specific validators is, in principle, externally observable over many rounds — but the current design gives no individual validator a way to confirm, for a single round, whether its own submission was included in the finalized result. For T6, stalling is bounded rather than prevented: a stalled round cannot exceed T_round before timing out, so the attack degrades liveness within a fixed window rather than indefinitely. Section IX discusses a multi-aggregator design that would close T4 and T6 completely rather than merely bounding them.

F. Replay Attack Prevention (T5)

An authentication message M_auth = pk_v// T carries only a validator's key and a timestamp — it contains no round identifier. Reusing a captured (pk_v, T, σ_auth) tuple in a later round fails for two independent reasons: the timestamp window Δ_auth bounds how long the message remains valid at all (§VA3), and the per-round pull restriction in Algorithm 2's check 3 rejects any pk_v that has already pulled a task for the current round. Once a round closes, any stale authentication attempt is rejected at Algorithm 2, step 13, before it can be processed further — closing T5 without requiring a separate nonce registry.

VII. Experimental Findings

This section evaluates the architecture's practical performance and the empirical strength of the quorum agreement guarantee discussed in Section VI, under a real multi-region deployment.

A. Deployment Setup

The protocol was deployed on the Solana devnet and evaluated over a 7-day period using five validator nodes across three continental regions: North America (2 nodes), Europe (2 nodes), and Asia (1 node). Six target URLs were monitored — three high-availability commercial services (SLA ≥ 99.9%) and three test services. The quorum threshold was Q = 3 and the round interval I = 5 minutes, yielding 2,016 rounds per target over the evaluation window. All validators ran on commodity cloud VMs (2 vCPU, 4 GB RAM) on AWS.

B. Attestation Latency and Commitment Cost

Round latency — from the validator's HTTP check to the confirmed on-ledger PDA write — was measured across all 2,016 rounds. Mean anchoring latency was 3.8 s (99th-percentile 5.4 s), well within the 120 s round timeout, content-store upload accounted for 55.3% of total anchoring time (2.1 s of 3.8 s mean). Mean ledger transaction fee per round finalization was 0.000025 SOL, excluding Round Summary Account rent (which is refunded on account closure), confirming the economic viability of high-frequency anchoring. No round failed to reach on-ledger confirmation during the evaluation.

TABLE II. Round Count and Anchoring Cost Across Subscription Intervals

Interval (min)	Rounds/Day	Cost/Day (SOL)	Cost/Month (SOL)
5	288	0.0072	0.216
10	144	0.0036	0.108
15	96	0.0024	0.072
30	48	0.0012	0.036
60	24	0.0006	0.018

Table II shows that anchoring cost is inversely proportional to the monitoring interval cost-efficiency is a deployment parameter, not a fixed property of the architecture.

C. Quorum Agreement Under Real-World Conditions

Across the three high-availability targets, quorum consensus (all Q ≥ 3 participating validators agreeing on is_up) was reached in 99.7% of rounds. Disagreement events — one validator reporting down while others reported up — correlated with regional network anomalies and occurred at a rate of 0.3% (about 6 of 2,016 rounds per target). Each of two deliberately induced blackouts on test services was caught within a single round, with uptime dropping to zero immediately, and detected within 1–2 minutes in the worst case — the window the aggregator waits for either full validator submission or timeout.

D. Independent Verification Performance

Algorithm 4 was executed against 200 randomly sampled historical rounds by an independent client with no access to the aggregator. All 200 rounds returned VALID, with a mean verification time of 78 ms per round (n = 5 validators, commodity laptop hardware), consistent with O(n) Ed25519 verification cost. Fig. 3 shows verification time scaling linearly

with validator count; 1,000 total signature verifications (200 rounds × 5 validators) completed in 15.6 s with no false positives or negatives — empirically confirming the independent verifiability guarantee argued in Section VI.

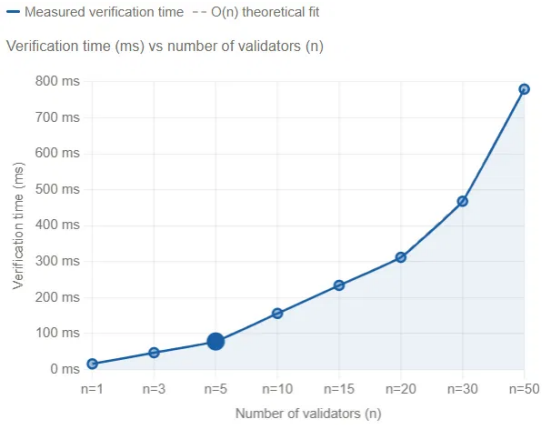

Fig. 3. Independent verification time per round as a function of validator count n, showing O(n) linear scaling consistent with the per-signature Ed25519 verification cost.

VIII. Security Posture Comparison

Table III compares the resulting security posture against centralized monitoring services along the dimensions most relevant to the threat model of Section II, rather than along general feature lines.

TABLE III. Security Posture: Centralized Monitoring vs. Proposed Architecture

Property	Centralized	Proposed
Trust model	Full provider trust	Cryptographic verification
Evidentiary integrity	No tamper evidence	SHA-256 + ledger commitment
Non-repudiation	None	Ed25519-signed submissions
Sybil cost	None	Capital cost per identity [Eq. (8)]
Independent verifiability	Not supported	Fully supported, O(n)
Insider-attack detectability	No	Yes — public audit trail
Residual trust required	Full (provider)	Bounded (§VI-E)
Operational complexity	Low	High

Traditional centralized monitoring systems require users to place complete trust in the service provider, which performs monitoring, stores the results, and generates compliance reports without independent verification. Consequently, there is no built-in mechanism to detect tampering, provide nonrepudiation, or enable third parties to verify the authenticity of reported results. Such systems also offer no inherent protection against Sybil attacks, lack transparency in detecting insider attacks, and rely entirely on the provider's integrity. Although centralized solutions are operationally simple and incur relatively low overhead, they introduce a single point of trust and potential failure.

In contrast, the proposed blockchain-based framework replaces implicit trust with cryptographic verification. Monitoring evidence is protected using SHA-256 hash commitments that are immutably recorded on the blockchain, while Ed25519 digital signatures provide non-repudiation by ensuring that validators cannot deny their submitted results. The framework discourages Sybil attacks by imposing an economic cost for each validator identity and enables independent verification of monitoring evidence with linear-time (O(n)) complexity. Furthermore, the public audit trail facilitates the detection of insider attacks and significantly reduces the amount of trust placed in any single entity. While the proposed approach introduces higher operational complexity due to cryptographic operations, blockchain transactions, and distributed consensus, it provides substantially stronger guarantees of integrity, transparency, accountability, and verifiability.

The Residual Trust Required row is the most consequential entry: centralized monitoring requires unconditional trust in the provider, whereas the proposed architecture reduces that requirement to a single, explicitly bounded assumption — a semi-honest aggregator that can at worst delay or selectively omit evidence within the bounds proved in Section VI-E, rather than fabricate or silently alter it. Section IX discusses removing even that bounded assumption.

IX. Conclusion and Future Work

This study formalized monitoring-data integrity as an adversarial security problem rather than an operational inconvenience, defined an explicit adversary model and threat taxonomy (Section II), and presented a cryptographic verification architecture — Ed25519- signed observations, quorum-based Byzantine-fault-tolerant aggregation, SHA-256 hash-committed ledger anchoring, and stake-bonded Sybil resistance — engineered against that adversary. Signature unforgeability was proved under the ECDLP/EUFCMA assumption, the capital cost of quorum capture was derived, commitment-immutability and hash-binding guarantees were established, and the residual attack surface that remains under a semi-honest aggregator — selective evidence inclusion and round stalling — was explicitly bounded rather than dismissed. A seven-day multi-region deployment validated the architecture's practicality, achieving 99.7% quorum agreement and zero verification errors across 200 independently re-checked historical rounds.

The most significant open threat is the semi-honest aggregator assumption itself. A multi-aggregator design, 9 in which several independent aggregators race to finalize each round and their outputs are cross-checked on the ledger, would remove the assumption that any single aggregator is even semi-honest, closing T4 and T6 entirely rather than bounding them. A second direction is shrinking the trusted computing base further by formally verifying the on-ledger program logic — for instance with a model checker or a Solana-specific static analyzer — since the protocol-level proofs in Section VI assume the commitment program is implemented exactly as specified, an implementation bug below that abstraction layer sits outside this study's threat model but not outside a real deployment's risk surface. Other directions include Merkle-root compaction of historical round summaries to retain on-ledger verifiability without per-round storage cost, and zero-

knowledge proofs of aggregate uptime that would let a client verify a result without downloading every individual signed submission.

More broadly, the pattern demonstrated here —independent signed observation, quorum-based aggregation, and hash-committed public anchoring — generalizes to other domains where a single reporting party currently self-attests to facts it has a financial or reputational interest in misrepresenting, including SLA compliance reporting, supply-chain provenance claims, and automated compliance audit logs.

REFERENCES

[1] Faiza Qazi, Daehan Kwak, Fiaz Gul Khan, Farman Ali, Sami Ullah Khan, Service Level Agreement in cloud computing: Taxonomy, prospects, and challenges, Internet of Things, Volume 25,2024,101126, ISSN2542-6605, https://doi.org /10.1016 /j.iot. 2024.101126.(https://www.sciencedirect.com/science/article/pii/S2542660524000684)

[2] Suhas et al., "Blockchain-Based Service Level Agreement Verification Framework", International Research Journal on Advanced Engineering and Management (IRJAEM), 4(06), pp.2343–2348m,2026. https://doi.org/10.47392/IRJAEM.2026.0336.

[3] A. Shostack, "Threat Modeling: Designing for Security," Wiley, 2014.

[4] R. Ross, M. Winstead, and M. McEvilley, "Engineering Trustworthy Secure Systems," NIST Special Publication 800-160, Vol. 1, Rev. 1, Nov. 2022.

[5] Shwethashree et al., "Multimedia Encryption Using Rubik's Cube-Inspired Algorithm: A Novel Approach to Audio and Image Security", EAI Endorsed Trans IoT, vol. 11, Mar. 2026.

[6] Pingdom, "Pingdom Website Monitoring," SolarWinds, 2024.

[7] UptimeRobot, "UptimeRobot Free Website Monitoring," 2024.

[8] Datadog, "Synthetic Monitoring," Datadog, Inc., 2024.

[9] E. Androulaki et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in Proc.13th EuroSys Conf., 2018, pp. 1–15.

[10] V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum White Paper, 2014.

[11] A. Yakovenko, "Solana: A New Architecture for a High-Performance Blockchain," Solana Labs White Paper, 2018.

[12] J. Benet, "IPFS — Content Addressed, Versioned, P2P File System," arXiv:1407.3561, 2014.

[13] Anchor Framework, "Anchor: A framework for Solana's Sealevel runtime," Coral, 2024.

[14] S. Ellis, A. Juels, and S. Nazarov, "Chainlink: A Decentralized Oracle Network," Chainlink White Paper, 2017.

[15] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," J. Cryptographic Engineering, vol. 2, no. 2, pp. 77–89, 2012.

[16] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Trans. Program. Lang. Syst., vol. 4, no. 3, pp. 382–401, 1982.