Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

NEITH-ACE: Authenticated Telemetry Completeness and Loss Attribution for Small-Spacecraft Flight Software

Mohamed El-Hadedy

mealy@cpp.edu

California State Polytechnic University

Research Article

Keywords:

Posted Date: November 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11690675/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

NEITH-ACE: Authenticated Telemetry Completeness and Loss Attribution for Small-Spacecraft Flight Software

Mohamed El-Hadedy[1,2,*]

[1]Department of Electrical and Computer Engineering, California State Polytechnic University, Pomona, CA, USA

[2]Coordinated Science Laboratory, University of Illinois Urbana-Champaign, Urbana, IL, USA

[*]Corresponding author: mealy@cpp.edu; also hadedy@illinois.edu ORCID: 0000-0002-3823-0712

Abstract

After an anomaly, a small spacecraft's telemetry is often the only evidence, yet an authentic record can still be incomplete. Existing integrity mechanisms, including the NEITH scheme extended here, cover only what reached the onboard logger (the integrity service). NEITH-ACE checks completeness against the source. Each capture adapter numbers its chunks and signs chained manifests of what it captured, under a key the integrity service never holds; every window commits an account of present, missing, failed, and late chunks; and the ground store signs receipts. All signatures use the hash-based eXtended Merkle Signature Scheme (XMSS). A fully compromised integrity service colluding with the store cannot certify an incomplete record as complete without forging a signature or finding a hash collision; with an honest integrity service, each missing chunk is traced to the last authenticated boundary it crossed. On a testbed with a live navigation receiver and laser scanner, one run repaired all 205 chunks a lossy link dropped and located the 11 stored chunks a cheating store altered. Against reimplementations of existing mechanisms, logger-trusting ones certified incomplete records as complete; source-signed schemes did not, but only NEITH-ACE also repaired losses and located custody failures, at a measured evidence cost of 10.4% of the telemetry. Inside NASA's F′ framework, replaying recorded data, the capture adapter sealed each manifest in a 55 ms median of its 10 s period. An investigator learns what is missing and where its evidence ends.

Keywords: telemetry completeness; tamper-evident logging; post-quantum signatures; XMSS; small spacecraft; flight software

1 Introduction

After an anomaly, the telemetry a small spacecraft returned is often the only record of what happened. Before anyone reads it, that record passes from the sensors through onboard flight software, across short and lossy ground passes, into an archive run by someone else. When a satellite-navigation (GNSS) trace shows a forty-second hole, or a laser-scanner (LiDAR) log stops just before the event, an investigator needs to know whether the data was lost on the link, dropped onboard, or deleted later, and whether someone removed exactly the interval that mattered.

Existing integrity mechanisms cannot make this distinction. Signatures, message authentication codes (MACs), and Merkle logs prove that the records present are authentic and unchanged, but they say nothing about records that never reached the onboard logger. The conference version of this work, NEITH, divides telemetry into *chunks*, the adapter's unit of capture such as a batch of GNSS sentences or a block of LiDAR returns, and commits each fixed time window of chunks to one Merkle root [1]. It shares the same limit: a chunk lost before the logger never enters a window, and the root commits an incomplete record with no sign of the loss. A compromised logger can also sign a shortened history that verifies perfectly. In this article the onboard logger is called the *integrity service*.

The problem is therefore to let a ground verifier, which trusts neither the link, nor the archive, nor necessarily the integrity service, decide whether a stored record holds every chunk the trusted capture adapters declared and, when one is missing, identify the last authenticated boundary it crossed: the adapter's signed manifest, the integrity service's committed record, or the store's signed receipt. For instance, the evidence for a chunk listed in a manifest but absent from every record ends at the adapter's manifest, and the evidence for a chunk the store signed for but can no longer produce ends at the store's receipt; neither tells who removed it or why. Four requirements make an answer useful. An incomplete record must never be certified complete, even when the integrity service that built it is compromised. Each missing chunk should be attributed to that boundary, not to a guessed physical cause, actor, or failure location. A gap that can still be closed should be closed, with evidence, rather than only reported. Finally, the evidence must fit a fixed deadline on small flight processors and use only hash-based and symmetric primitives, whose quantum resistance then depends on the chosen parameters.

NEITH-ACE meets these requirements by counting at the source and keeping signed evidence at each handoff. The capture adapter, the first programmable point a sample passes, numbers every chunk and periodically signs a chained *manifest* of what it produced, under a key the integrity service never holds. The integrity service commits, per window, an account of every chunk that arrived, failed authentication, or was missing, and asks the adapter to resend what is missing; the ground archive, the *store*, signs a receipt for every chunk it accepts. The verifier judges completeness against the manifests rather than the integrity service's word, and the records and receipts give the last authenticated boundary each missing chunk crossed.

Source signatures alone are not new. Logger-side Merkle logs such as Certificate Transparency (CT) [2] cover only what was submitted to them, and the space-link security (SDLS) and file-delivery (CFDP) protocols of the Consultative Committee for Space Data Systems [3, 4] protect one hop but not the stored copy. Signed syslog [5] and DASLog [6] have each source sign its own messages, so both detect an integrity service that drops or fabricates records. Neither attributes a missing record to the last authenticated boundary it crossed, and neither repairs a recoverable gap with evidence. What NEITH-ACE adds is the combination: source-signed manifests together with window-level accounting, evidence-guided repair, signed custody receipts, and attribution, all with hash-based signatures and with the capture side running in flight software.

The article makes four contributions. First, it gives a construction in which capture adapters sign chained manifests, each window commits a per-sensor account of present, missing, failed, and late chunks, the store signs receipts, and recoverable gaps are re-requested and repaired (Section 3). Second, it proves four guarantees, the central one being that a fully compromised integrity service, even colluding with the store, cannot pass an incomplete record off as complete without forging an XMSS signature or finding a hash collision (Sections 2 and 4). Third, it derives closed-form bounds that turn the window deadline into a maximum chunk rate and an XMSS key lifetime, so a deployment can be sized before launch (Section 5). Fourth, it evaluates the scheme on a three-machine testbed with live GNSS and LiDAR, under an attack injector that includes a compromised integrity service, and against reimplementations of five existing mechanisms and a freshness variant of one in verdicts and evidence cost, with its capacity measured on a Raspberry Pi 4 and the capture adapter also running inside F′, NASA JPL's open-source flight-software framework (Section 6). All four are new relative to the conference version [1], from which only the window map and the no-stutter rule of Section 2 carry over.

2 Background and Threat Model

Five kinds of party take part, shown in Fig. 1: one capture adapter per sensor, the integrity service, the store, the verifier, and the adversary. On a spacecraft the capture adapters and the integrity service run onboard, the adapters in a partition the service cannot read, such as a secure element, FPGA fabric, or an isolated core (the testbed of Section 6 approximates this with separate processes and does not enforce it); the store is the ground archive, and the verifier is a ground station or auditing authority that sees roots, headers, records, checkpoints, and whatever

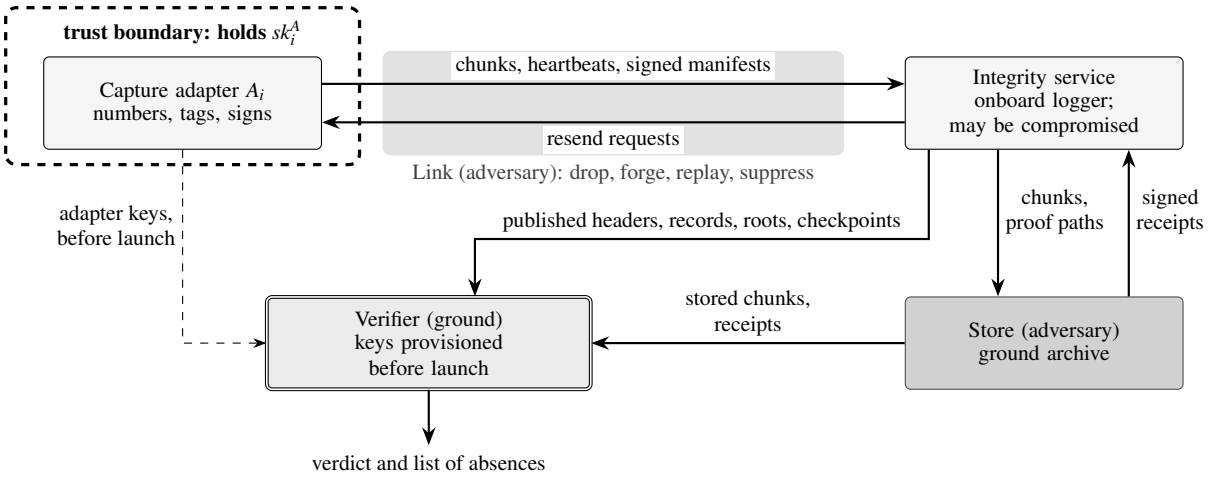

Fig. 1 Parties, trust, and evidence flow; dark boxes are adversarial, the dashed rectangle is the adapter's trust boundary, the double border marks the verifier, and the dashed arrow shows keys provisioned before launch

Table 1 Who is trusted in each case

Party	Honest service	Compromised service
Capture adapters, keys K_i, sk_i^A	trusted	trusted; sk_i^A never leaves the adapter
Integrity service, keys K_i, sk	trusted	adversarial; its gap classes and arrival times are only claims
Link	adversarial	adversarial
Store, key sk_{st}	untrusted; its receipts bind custody	untrusted; its receipts bind custody only if it does not collude with the service
Verifier, keys provisioned before launch	trusted	trusted

the store returns.

NEITH, which NEITH-ACE extends, measures time at the integrity service from the start τ_0 of an epoch e, one numbered run of the pipeline, and, for a window length $\Delta > 0$, maps ingestion time $\tau \geq \tau_0$ to the window index

$$t(\tau) = \left\lfloor \frac{\tau - \tau_0}{\Delta} \right\rfloor, \tag{1}$$

so window t closes at $T_t = \tau_0 + (t+1)\Delta$. The chunks of a window are reduced by a domain-separated binary tree into one 32-byte root, and an inclusion proof is the $\lceil \log_2 N_t \rceil$ siblings on a leaf's path [1]. The hash H is any collision-resistant function with 32-byte output, such as SHAKE128 [7] or Ascon-XOF128 [8]. With E_t the time the service emits root t (distinct from when a verifier receives it), the no-stutter rule

$$E_t - T_t \leq \Delta \quad \text{for all } t \tag{2}$$

keeps lateness from accumulating across windows. NEITH-ACE keeps both and changes what a root commits.

Every hop before the verifier is open to the adversary $\mathscr{A}$, a probabilistic polynomial-time party, possibly quantum, that observes all public values and traffic and may act in four places at once. On the *link* between adapters and service, in both directions, it drops, delays, duplicates, reorders, replays, modifies, or injects chunks, heartbeats, manifests, and resend requests. On *publication* it withholds, reorders, truncates, or substitutes headers, records, roots, and checkpoints. In *custody* it deletes, corrupts, or substitutes stored chunks at any later time and refuses or delays receipts. If the service is also compromised, the adversary holds every MAC key K_i and the checkpoint key sk, computes records, roots, and checkpoints however it likes, and may collude with the store.

What remains trustworthy depends on the case, as Table 1 summarizes. With an *honest service*, the trusted parties are the adapters, the service, and the verifier with its keys, as in logs that rely on a trusted logger [9]. With a *compromised service*, trust shrinks to the adapters, their manifest keys sk_i^A, and the verifier, which holds the

configuration and every public key, provisioned out of band before launch; the service's gap classes and arrival times then become claims. The store is untrusted in both cases, and receipts attribute custody loss only while the store does not collude with the service. No stateful key reuses an index or signs more than its 2^h messages, and an adapter never reuses a sequence number, counter, or manifest index within an epoch. Adapter compromise and false measurements such as GNSS spoofing [10] lie outside the model.

The cryptographic assumptions are standard. H is collision resistant, enc is injective and prefix-free, MAC is unforgeable under chosen messages with verification queries, and the signature schemes Σ (checkpoints), Σ^A (manifests), and Σ_{st} (receipts) are existentially unforgeable under chosen-message attack (EUF-CMA); all three are XMSS-SHAKE256, whose security rests only on hash properties and holds while no one-time index repeats [11, 12, 13]. Online Resource 1 gives the margins these choices leave against a quantum adversary.

Timing assumptions are few. Service and verifier share the schedule T_t. The service closes window t after a grace delay $0 \le \gamma < \Delta$ and assigns any later arrival to a later window, where it resolves the gap. Adapters need no synchronized clock: they send a heartbeat every Δ_h seconds and a manifest every Δ_m seconds, a sensor without a heartbeat for $\kappa\Delta_h$ is flagged stale, and the service signs a checkpoint every k windows. In checkpoint windows E_t also covers σ_t, and the publication allowance must cover the honest delay, $\theta \ge \max_t(E_t - T_t - \gamma) + d_{\text{pub}} + \varepsilon$, where d_{pub} bounds honest delivery and ε the clock skew between service and verifier; otherwise a slow but honest service would be reported as withholding. The deadline and the allowance hold only for bounded arrivals: a flooding adversary can force a WITHHELD verdict, which is a denial of service, not a false claim of completeness.

Confidentiality and availability are not goals: cutting links stops the pipeline but cannot make it look healthy.

3 NEITH-ACE Design

A NEITH-ACE root commits three things: what reached the integrity service, an authenticated account of what should have reached it, and the previous root. Fig. 1 in Section 2 shows how that evidence is gathered. Along the top, each capture adapter sends its chunks, tags, heartbeats, and signed manifests across the link to the integrity service, which can ask for lost chunks to be resent. The service hands each window's chunks to the store, which answers with signed receipts, and publishes headers, roots, and checkpoints. The verifier combines these published values with the adapters' manifests, whose keys it received before launch (dashed arrow), and with the store's copy and receipts, and reaches a verdict; the possible verdicts are explained at the end of this section, once the evidence behind them has been described. Algorithm 1 lists what each honest party does, with one block per party: lines 1–4 run on the adapter, lines 5–10 on the integrity service once per window, and line 11 on the store. Read top to bottom, the algorithm follows one chunk from capture to storage: the adapter declares it, the integrity service accounts for it, and the store acknowledges custody.

Each sensor $i \in \{1, \ldots, m\}$ has a capture adapter A_i, the first programmable point on its path. The adapter shares a MAC key K_i with the integrity service and holds its own manifest key sk_i^A; the service holds the checkpoint key sk, and the store holds a receipt key sk_{st}. All three signing roles use XMSS [11, 12], H is the 32-byte hash of Section 2, MAC is the standardized KMAC128 [14] or, in the F′ adapter, Ascon-CXOF128 [8] keyed by a key prefix, whose unforgeability is an assumption, motivated by keyed-sponge results [15] but not proven or standardized, and enc is an injective, length-prefixed encoding. Every protocol hash starts with its own one-byte domain tag, listed in Online Resource 1.

The adapter first tells the service what exists, without a clock shared with the sensors. It numbers its chunks $s = 0, 1, \ldots$ within epoch e and tags each one (line 1), and every Δ_h seconds it sends a heartbeat carrying $\hat{s}$, the highest number sent so far, and a strictly increasing counter u that stops old heartbeats from being replayed (line 2). These tags authenticate chunks only to the service, which holds K_i and, if compromised, could invent a matching history. The adapter therefore also signs, once per Δ_m, a manifest listing the digest of every chunk captured since the previous one (line 3). Each manifest carries the epoch, the sensor number, and the hash of its predecessor, and

Algorithm 1 Evidence at the adapter, commitment at the service, and custody at the store

Adapter A_i, epoch e; initially $s, \hat{s} \leftarrow -1$, $u \leftarrow 1$, $v \leftarrow 1$, $\mu_0 \leftarrow H(\texttt{0x09} \| \mathsf{enc}(e,i))$, open list $\Pi \leftarrow \langle\rangle$

1: on chunk c: $s \leftarrow s+1$; $\hat{s} \leftarrow s$; send (i,s,c,α) with $\alpha = \mathsf{MAC}_{K_i}(\mathsf{enc}(\texttt{"chunk"}, e, i, s) \| c)$; keep it in the buffer; append $(s, H(c))$ to Π

2: every Δ_h: send heartbeat $(i, \hat{s}, u, \mathsf{MAC}_{K_i}(\mathsf{enc}(\texttt{"hb"}, e, i, \hat{s}, u)))$; $u \leftarrow u+1$

3: every Δ_m, and once more with $fin = 1$ at stop: $\mu_v \leftarrow H(\texttt{0x0a} \| \mathsf{enc}(\texttt{"mf"}, e, i, v, \hat{s}, \mu_{v-1}, fin, \Pi))$; send manifest $(v, \hat{s}, \mu_{v-1}, fin, \Pi, \mathsf{Sign}_{sk_i^A}(\mu_v))$; $v \leftarrow v+1$; $\Pi \leftarrow \langle\rangle$

4: on a request for chunk s or manifest v: resend the identical message if still buffered

Integrity service, window t, at $T_t + \gamma$

5: each arrival in window t: bad tag on an unresolved number $\rightarrow \mathscr{G}^{\text{auth}}$; duplicate $(i,s) \rightarrow$ drop; otherwise leaf $H(\texttt{0x00} \| \mathsf{enc}(e,t,i,s) \| c)$

6: per sensor: record $\rho_{i,t} = (i, a, b, \mathscr{G}^{\text{miss}}, \mathscr{G}^{\text{auth}}, \mathscr{L}, \mathscr{Q}, r, \hat{s}, u, \phi, \mathscr{M})$

7: $\Lambda_t \leftarrow$ binary tree over the N_t leaves in (i,s) order, nodes $H(\texttt{0x01} \| x_L \| x_R)$, an odd node paired with the constant padding $H(\texttt{0x03})$, a single leaf its own root, empty tree $H(\texttt{0x04})$; $D_t \leftarrow H(\texttt{0x02} \| \mathsf{enc}(\rho_{1,t}, \ldots, \rho_{m,t}))$

8: $R_t \leftarrow H(\texttt{0x05} \| \mathsf{enc}(e,t,N_t) \| \Lambda_t \| D_t \| R_{t-1})$, with $R_{-1} = H(\texttt{0x06} \| \mathsf{enc}(e, \mathsf{cfg}, pk, \langle pk_i^A \rangle, pk_{\text{st}}))$

9: publish $h_t = (t, N_t, \Lambda_t, D_t)$, the records, and R_t; forward the chunks, h_t, and their proof paths to the store; request unresolved gaps and missing manifests

10: if $t \equiv k-1 \pmod{k}$ or at shutdown: persist the XMSS index, then publish $\sigma_t = \mathsf{Sign}_{sk}(H(\texttt{0x07} \| \mathsf{enc}(e,t,end) \| R_t))$, with $end = 1$ only at shutdown

Store, window t

11: check each chunk against Λ_t, persist it, set its bit in β_t; return $q_t = \mathsf{Sign}_{sk_{\text{st}}}(\mathsf{enc}(\texttt{"rcpt"}, e, t, N_t) \| \Lambda_t \| \beta_t)$

Table 2 Fields of the record $\rho_{i,t}$ of sensor i in window t, and the rule each satisfies

Field	Meaning	Rule
$[a,b]$	numbers this window accounts for	a follows the previous window's b; b is the highest number received or declared
$\mathscr{G}^{\text{miss}}$	numbers never received	with $\mathscr{G}^{\text{auth}}$ and the leaves, covers $[a,b]$ exactly once
$\mathscr{G}^{\text{auth}}$	numbers that arrived only with bad tags	a later bad-tag arrival may move a number here from $\mathscr{G}^{\text{miss}}$
$\mathscr{L}$	late arrivals	each closes a gap recorded in an earlier window
$\mathscr{Q}, r$	receipts that arrived, and their digest	a receipt enters the window it arrives in, never the root it acknowledges
$(\hat{s}, u)$	latest heartbeat: highest number sent, counter	u strictly increases
ϕ	staleness flag	raised after $\kappa\Delta_h$ without a heartbeat
$\mathscr{M}$	manifests that arrived	committed in chain order, none skipped

the last one carries $fin = 1$, so a manifest cannot move between epochs or sensors and a skipped one is visible; the verifier holds pk_i^A itself, so the service cannot substitute its own. The verifier judges completeness against these manifests, that is, against the source.

Using these declarations, the service accounts for every number in each window. It assigns each arrival to window t by its own clock, records a bad tag as an authentication failure, drops duplicates, and turns every other chunk into a leaf (line 5), and at $T_t + \gamma$ it writes one record per sensor (line 6). Table 2 lists the record's fields and their rules, which make every number in $[a,b]$ either a leaf or a gap, exactly once (all symbols: Online Resource 1).

A recorded gap can often still be closed, because the adapter may still hold the chunk. Fig. 2 follows one window in which this happens; time runs downward and each column is one party. The adapter sends chunk s, but chunk $s+1$ is lost on the link (cross). The next heartbeat still announces $\hat{s} = s+1$ and the manifest lists its digest, so when the window closes the service records $s+1$ as missing, adds the manifest to the record, and commits the root. After the window the service asks for $s+1$, the adapter resends the identical tagged chunk from its buffer of the last B chunks (line 4), and the late arrival resolves the gap in the next record; the lower part of the figure shows the custody steps described below. A valid resend is treated as a late arrival, so repair needs no new verification rule, and it must match the digest the manifest already fixed; suppressing requests leaves the verdict at ACCOUNTED.

The leaves and the records are then bound into one root that also commits the leaf count, the window index, and the previous root (lines 7–8). The leaf count fixes the tree shape, the index prevents replay into another slot, and the chain reaches back to a genesis that fixes the configuration, including the declaration mode, and every public key; the verifier holds these itself, provisioned before launch, and rejects a genesis that does not match.

A committed root does not cover what happens to the chunks afterwards. To separate loss in custody from loss before it, the store checks every committed chunk against Λ_t, persists it, and signs one receipt per window over a bitmap β_t of the positions it accepted (line 11); a set bit binds the store to that exact content. A receipt is added

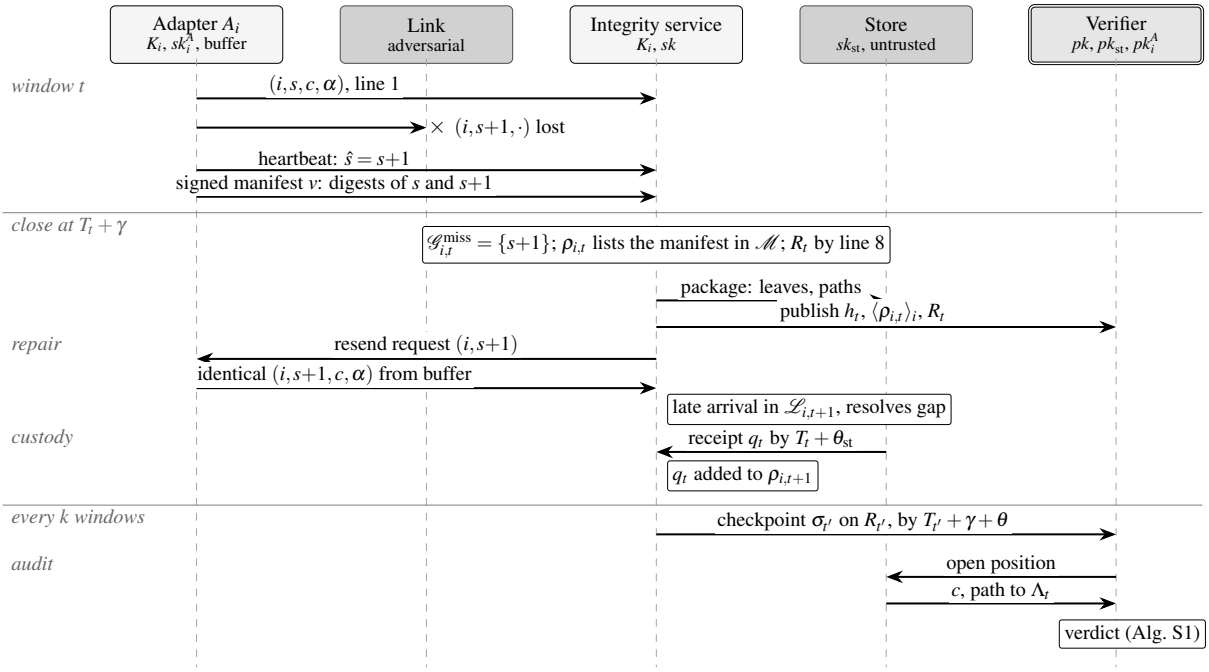

Fig. 2 One protocol window: a lost chunk is declared, recorded as a gap, and repaired. Time runs downward and each column is a party (dark: adversarial); $\times$ marks a chunk lost on the link; line numbers refer to Algorithm 1, and Alg. S1 is in Online Resource 1

to the record of the window in which it arrives, so it never rewrites the root it acknowledges, and a position still unacknowledged after θ_{st} counts as absent at the handoff. Because each root contains its predecessor, the service signs a checkpoint only every k windows and at shutdown, persisting the XMSS state first (line 10); rollback resistance after a restart is assumed.

The verifier (Algorithm S1 in Online Resource 1) checks the genesis against its own keys, recomputes the chain, checks checkpoints, manifests, and receipts, and opens every committed position in the store, whether or not it was receipted. A record that commits a number twice, lists a gap outside its interval, or commits a number the adapter never declared, below its declared frontier or after its final manifest, is INVALID; beyond the frontier it is STALE. Table 3 lists the possible verdicts. The verifier stops at the first INVALID or WITHHELD finding it meets while walking the history; otherwise it returns the first of the remaining rows whose condition holds, so a history with both a forged leaf and a missing chunk is INVALID, not ACCOUNTED. The service records manifests in chain order, holding a later one until a re-requested earlier one arrives, so an honest record never skips one. An ACCOUNTED verdict also labels each absence with the last place it was accounted for: INGESTABSENT (declared by the adapter, never ingested by the service), INGESTAUTHFAIL (arrived only with bad tags), HANDOFFABSENT (committed by the service, never receipted by the store), or CUSTODYLOSS (receipted by the store, no longer producible); an opening that fails its proof counts as a custody loss, never as an invalid history, so a dishonest store cannot invalidate an honest record.

4 Security Analysis

The guarantees below reduce to standard assumptions, with Adv_H^{cr} the advantage of finding a collision in H and $\text{Adv}_X^{\text{euf}}$ that of forging against $X \in \{\text{MAC}, \Sigma, \Sigma^A, \Sigma_{st}\}$; for XMSS the bounds hold only while no one-time index repeats [16]. The statements are given here in compact form; Online Resource 1 states and proves them in full.

A root must first fix exactly one content for its window. Call an *opening* of R_t everything needed to recompute it: the previous root, the header, the records, and the N_t chunks behind the leaves.

Table 3 Verdicts; the last five in the order the verifier tests them.

Verdict	Returned when	What the investigator learns
INVALID	a checkpoint or manifest signature, chain link, root, or genesis fails verification, a committed record breaks the accounting rules above, or a verified opening contradicts its declared digest	the history itself cannot be trusted
WITHHELD	a checkpoint is missing past its deadline, or a record skips a manifest	evidence was suppressed or delayed
STALE	a sensor's heartbeats stopped, or committed chunks run past its last manifest	the source fell silent or is not yet confirmed
ACCOUNTED	some declared chunk is absent, or a committed one was never receipted	what is missing, and where each absence was last accounted for
PREFIX	nothing is absent, but a final manifest is missing, windows await a checkpoint, or positions are not yet opened or their receipts not yet due	complete as far as it goes
REPAIRED	nothing is absent, and some gap was closed by an authenticated late arrival	complete, with the repairs listed
COMPLETE	nothing is absent, and no gap was ever recorded	complete

Theorem 1 (Window binding)**.** *Any adversary that opens the same root R_t in two different ways yields a collision in H; its success probability is at most* $\mathrm{Adv}^{\mathrm{cr}}_{H}$.

The first differing input to a tagged hash is the collision: D_t fixes the records, with their gap sets and late arrivals, and N_t fixes the tree shape. Because each root contains its predecessor, binding one window extends to the whole history behind a checkpoint, provided the checkpoint signer is honest; a compromised service holding *sk* could sign another chain, which is why Theorem 4 does not rely on the next result.

Theorem 2 (History integrity)**.** *If the checkpoint signer is honest, its XMSS state is never rolled back, and no epoch number is reused under its key, an adversary that presents, under a valid checkpoint for window t^*, a sequence of openings different from the one the service committed succeeds with probability at most* $\mathrm{Adv}^{\mathrm{euf}}_{\Sigma} + \mathrm{Adv}^{\mathrm{cr}}_{H}$.

An unsigned checkpoint is a forgery; otherwise the first differing window, which the reduction finds directly, gives two openings of one root. Deleting, inserting, reordering, and substituting windows are all covered.

A fixed history is not yet a complete one. With an honest service, nothing an adapter declared can disappear unnoticed. Let $\hat{s}^*$ be the largest sequence number declared by a heartbeat, or carried by a validly tagged chunk, in a history verified through checkpoint t^*.

Theorem 3 (Declared completeness)**.** *If adapter A_i and the service are honest, K_i is secret, and the conditions of Theorem 2 hold, then, except with probability* $\mathrm{Adv}^{\mathrm{euf}}_{\mathsf{MAC}} + \mathrm{Adv}^{\mathrm{euf}}_{\Sigma} + \mathrm{Adv}^{\mathrm{cr}}_{H}$*, every $s \le \hat{s}^*$ ends, in the reconciled history, as exactly one leaf carrying the payload that A_i tagged or as an unresolved gap, never both; the reconciled history is the one obtained by applying each later repair to the gap it closes.*

The records cover the declared range without overlap, and an earlier record may list a gap that a later one repairs; a late arrival can only resolve an existing gap, and a foreign payload needs a MAC forgery. No custody scheme can detect the loss of chunks produced after the last declaration; staleness limits that interval to $\kappa\Delta_h$ plus one window.

Both results so far trust the service. A lying service holds every K_i and *sk*, so it can forge heartbeats, fabricate leaves, and sign a history that calls itself complete; what it cannot do is forge the adapter's manifests, and that is enough. Let $\mathscr{E}$ be the sequence numbers declared in A_i's manifest chain, closed by a final manifest ($fin = 1$), up to the scope of the audit.

Theorem 4 (No false completeness)**.** *Let A_i be honest, with a secret manifest key whose XMSS state never repeats an index, and let the verifier hold the configuration and pk^A_i from provisioning and have access to the store. Let the service be arbitrarily malicious and possibly colluding with the store. Except with probability* $\mathrm{Adv}^{\mathrm{euf}}_{\Sigma^A} + \mathrm{Adv}^{\mathrm{cr}}_{H}$*, the verifier returns* COMPLETE *or* REPAIRED *only if a final manifest of A_i lies inside the checkpointed scope, the committed leaves of sensor i are exactly $\mathscr{E}$, each once, and the store returns for every $s \in \mathscr{E}$ a payload with the digest A_i signed. No dropped, withheld, fabricated, duplicated, or altered chunk can be certified complete.*

In other words, the integrity service may forge its own records and checkpoints, but it cannot add an entry to, or remove one from, the adapter's signed declaration without the verifier detecting it. $\mathscr{E}$ comes from a chain the service can neither forge nor fork. Every way of departing from it ends in a verdict below REPAIRED in Table 3, and a substitute payload with a signed digest is a collision, whoever supplies it. Under a compromised service the difference between REPAIRED and COMPLETE is the service's claim; only the completeness of the declared record is guaranteed.

The same split governs attribution. With an honest service the four absence classes are disjoint, and CUSTODYLOSS also rests on the store's own signature. With a compromised service the ingestion classes and HANDOFFABSENT become the service's claims; CUSTODYLOSS still rests on the store's signature as long as the store does not collude, and Theorem 4 holds either way. Finally, suppressed publication cannot go unnoticed for long: a missing checkpoint at $T_t + \gamma + \theta$ marks every window since the previous one WITHHELD, so suppression is detected within $k\Delta + \gamma + \theta$ of the start of the earliest withheld window.

5 Overhead Model

A deployment must know before launch whether its sensors fit the deadline and how long its keys last. Window t receives n_t chunks of size L and has $N_t \le n_t$ leaves. With P workers, per-chunk costs $c_{\mathsf{MAC}}(L)$ and $c_H(L)$, node-hash cost c_{n}, per-sensor close cost c_ρ (heartbeats, manifests, record), root cost c_{root}, and signature cost c_{sig}, all in seconds, and $I_t = 1$ in checkpoint windows (0 otherwise), closing the window costs

$$W_t \le n_t\, c_{\mathsf{MAC}}(L) + N_t\, c_H(L) + \big(N_t - 1 + \lceil \log_2 N_t \rceil\big)\, c_{\mathrm{n}} + m\, c_\rho + c_{\mathrm{root}} + I_t\, c_{\mathrm{sig}} \tag{3}$$

for $N_t \ge 1$, the logarithmic term covering the at most one padding node per tree level; averaged over a checkpoint period the signature term is c_{sig}/k. Three assumptions turn this work into a deadline. Tag checks, leaf hashes, and node hashes run on $P \ge 1$ greedy workers (an idealized count, so the result is a sizing bound; Section 6 measures the actual capacity), so they finish within their total cost divided by P plus the longest chain of dependent steps, which is one tag check, one leaf hash, and $\lceil \log_2 N_t \rceil$ node hashes. Records, the root, and the signature then run one after another, with storage writes counted inside c_{sig}. Finally, no work carries over from window $t-1$; this holds by induction as long as the bound below stays under Δ. Under these assumptions,

$$E_t - T_t \;\le\; \gamma + \frac{n_t\,(c_{\mathsf{MAC}} + c_H + c_{\mathrm{n}})}{P} + c_{\mathsf{MAC}} + c_H + 2\lceil \log_2 N_t \rceil\, c_{\mathrm{n}} + m\, c_\rho + c_{\mathrm{root}} + c_{\mathrm{sig}}. \tag{4}$$

Chunk rate trades directly against per-chunk cost. Write (4) as $E_t - T_t \le \gamma + \lambda\Delta C/P + S$ with $n_t \le \lambda\Delta$, $C = c_{\mathsf{MAC}} + c_H + c_{\mathrm{n}}$, and $S = c_{\mathsf{MAC}} + c_H + 2\lceil \log_2 N_{\max} \rceil c_{\mathrm{n}} + m c_\rho + c_{\mathrm{root}} + c_{\mathrm{sig}}$ the remaining terms at a leaf bound $N_{\max} \ge \lambda\Delta$, since late arrivals are already counted in n_t. When $\Delta > \gamma + S$, the no-stutter rule (2) holds for every rate up to

$$\lambda_{\max} \;=\; \frac{P\,(\Delta - \gamma - S)}{\Delta C} \tag{5}$$

chunks per second. Because n_t counts resends, duplicates, and forgeries, a flooded link can exceed λ. The deadline governs roots only; receipts are asynchronous and judged against $T_t + \theta_{\mathrm{st}}$.

Key lifetime trades against checkpoint frequency. An XMSS tree of height h signs 2^h messages, so a checkpoint key lasts $\Omega = 2^h k\Delta$ seconds, less one index per orderly shutdown. At $\Delta = 10\,\mathrm{s}$ and $k = 6$, $h = 16$ lasts about 45 days and $h = 20$ about two years, at under 50 B/s of checkpoint traffic. A shorter withholding delay $k\Delta + \gamma + \theta$ costs lifetime and bandwidth. The store signs every window and, with $\Delta_m = \Delta$, each adapter signs once per window as well, so at $h = 20$ their keys last 121 days and bound the deployment unless receipts are batched or multi-tree XMSS^{MT} [11] is used.

6 Evaluation

The evaluation has three parts: a live end-to-end run of the pipeline with host-process adapters, an attack injector and a comparison with existing mechanisms in simulated time, and a separate run of the capture adapter inside F′ on replayed data. The testbed uses three machines under separate control. A ClusterHAT 2.5 carries a Raspberry Pi 4 host, which runs the capture adapters with a GNSS receiver (u-blox) and a LiDAR (Youyeetoo) attached, and four Pi Zero 2 W nodes, one of which runs the integrity service over a USB-gadget network hop; the board also hosts a perception runtime [17]. A separate AMD Zynq-7000 PYNQ board (dual-core 32-bit ARMv7) on a WireGuard overlay network runs the store and, as a separate process, the verifier. Parameters are $\Delta = 10\,\text{s}$, $\gamma = 0.5\,\text{s}$, $\Delta_h = 1\,\text{s}$, $\kappa = 3$, $k = 6$, $\theta = 2\,\text{s}$, $\theta_{\text{st}} = 10\,\text{s}$, $\Delta_m = 10\,\text{s}$, and $L \leq 8192$ bytes. The live pipeline uses SHAKE128, KMAC128, and XMSS-SHAKE256 with $h = 10$ (the xmss-reference code with its fast tree traversal) for all three signing roles, enough for one epoch; a mission needs taller trees, sized per role in Section 5. Online Resource 1 gives every setting needed to reproduce the results.

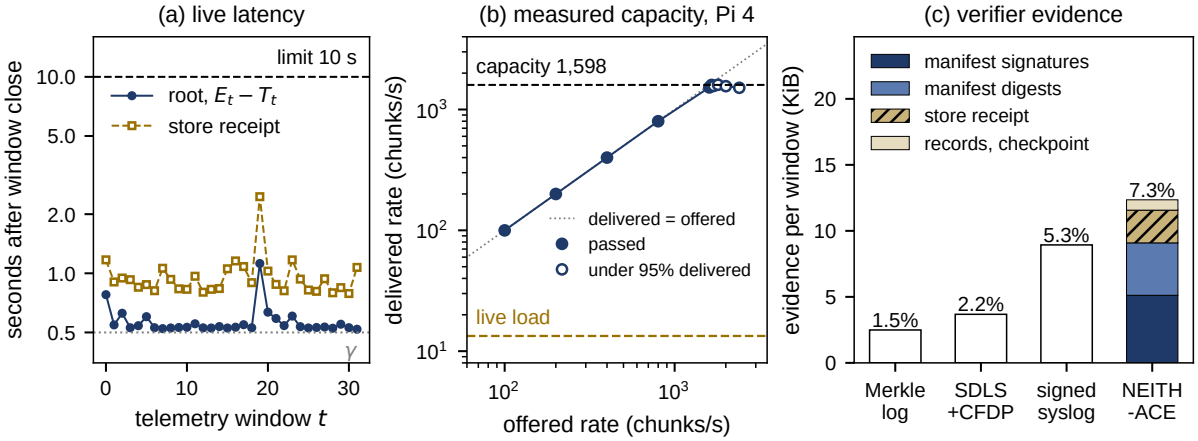

Fig. 3 (a) Root (circles) and store-receipt (squares) latency per telemetry window, against the 10 s limit (dashed) and γ (dotted). (b) Measured delivered rate against offered rate on the Pi 4 (filled: sustained; open: under 95% delivered); dashed lines mark the capacity and the live load. (c) Format-model evidence per window, percent of telemetry above each bar (NEITH-ACE's measured encoding: 10.4%); DASLog's ledger has no per-window size

On the Pi 4, XMSS-SHAKE256 at $h = 10$ generates a key in 4.3 s (274 s at $h = 16$), signs in a median 30.2 ms and at worst 41.1 ms, including the durable state write, and verifies in 2.3 ms; KMAC128 tags an 8 KiB chunk in 114 µs and SHAKE128 hashes it in 48 µs. Fig. 3(b) measures capacity directly. The whole pipeline, two adapters signing manifests, the integrity service, and the store with its durable writes, ran on the Pi 4 at rising offered rates of 8 KiB chunks for four telemetry windows each, with a checkpoint in every window ($k = 1$, the worst case). A rate counts as sustained when the verifier returns COMPLETE, every window meets (2), and the adapters deliver at least 95% of the offered rate. The pipeline sustained 1598 chunks/s (13.1 MB/s), about 120 times the live load of 13.4 chunks/s (dashed, resends included). Beyond that the delivered rate levels off between 1510 and 1600 chunks/s, because the adapters share the four cores with the service and the store; every run still ended COMPLETE, and even at an offered 2400 chunks/s each root came at most 4.4 s after its window closed. For other processors, (5) gives a first sizing from microbenchmark costs.

In the live run, over thirty-two telemetry windows (then a drain window for stragglers and a settlement window for receipts), the adapters captured 3416 GNSS and 654 LiDAR chunks and issued 33 signed manifests each. Fig. 3(a) shows how long after each window's close its root (circles) and store receipt (squares) appeared, against the 10 s limit (dashed) and the grace delay γ (dotted); the drain and settlement windows, both under 1 s, are omitted. The median root came 0.53 s after the close, almost all of it γ, and the slowest 1.12 s, in the stalled window 19; receipts took a median 0.90 s and at worst 2.45 s. The link dropped 5% of chunks; the service requested each gap,

and all 205 dropped chunks were resent and recovered, so the verifier's repair set equalled the injected loss. In the same run the ARMv7 store was made to cheat: it signed correct receipts and then, every third window, altered the oldest acknowledged GNSS chunk it still held intact. The verdict was ACCOUNTED, with exactly those eleven chunks as CUSTODYLOSS and no other absence, and verification took 7.6 s for the epoch. Here the verifier took the configuration and public keys from the honest integrity service's evidence package rather than from separate provisioning, so the live run does not test Theorem 4; the injector fixes the verifier's keys independently.

An attack injector tests the full threat model: missions of eleven or twelve windows with two sensors at the conference paper's rates (5.5 and 1.4 chunks/s), one attack per mission, real cryptography, and simulated time. Its 44 scenarios, listed in Online Resource 1, cover the link, publication, custody, and service positions, eight trials each, and check the exact verdict and, where the attack fixes one, the exact absence or repair set; all pass. The subtlest attacks, a service that fills a gap listed outside its record's interval with a fabricated or duplicated chunk, or forges a chunk whose receipt is not yet due, return INVALID because every committed position is opened. As an ablation, with declarations authenticated only by the MAC, a service that fabricates chunks or hides the newest windows gets COMPLETE, and one that forges an unreceipted last window gets REPAIRED; with adapter-signed manifests these attacks return INVALID, STALE, and INVALID, as Theorem 4 predicts.

Table 4 Verdicts on identical injected missions, twenty per cell.

				Compromised service	
Mechanism	Benign	Link loss	Store deletes	fabricates	hides newest
Merkle log (CT, history trees)	C	C*	D	C*	C*
Nitro (MAC combiner)	C	C*	D	C*	C*
SDLS + CFDP	C	R	n/a	C*	C*
Signed syslog (RFC 5848)	C	D	D	D	C*
Signed syslog + freshness	C	D	D	D	S
DASLog (source-signed chains)	C	D	D	D	D
NEITH-ACE	C	R	A	I	S/P

C complete; R repaired; D incomplete, loss not attributed to a step; A incomplete, each absence attributed to its last authenticated boundary; I invalid; S/P stale or prefix; n/a outside the mechanism's scope; * false claim of completeness.

Table 4 runs reimplementations of existing mechanisms on the same injected missions. The baselines are not deployments of the original systems: each reimplements the mechanism's completeness logic for this comparison and sees only the evidence that mechanism would hold. Every cell but one was the same in all twenty missions (NEITH-ACE returned STALE in 18 and PREFIX in 2 when the newest windows were hidden), so the table records structural outcomes, not rates. They are a logger-side Merkle log [2, 18]; Nitro's logger-side MAC combiner [19]; SDLS frame authentication with CFDP class-2 repair on the hop it protects [3, 4]; signed syslog [5], also with a verifier that knows the mission end and flags a source whose last block is too old; and DASLog's source-signed chains closed by a final message [6], reimplemented with the adapters' manifests and an honest ledger. Mechanisms that trust the integrity service, or cover one hop, certify incomplete records as complete; signed syslog with freshness and DASLog do not, because their source signatures detect every attack in the table. SDLS with CFDP also repairs the link loss, but only on the hop it protects, and it certifies the compromised service's records. Only NEITH-ACE both refuses a false COMPLETE and repairs the loss, and it alone attributes the store's deletions to the store's receipt.

This costs more evidence, as Fig. 3(c) shows. Each bar is what a verifier must receive per window besides the telemetry, counted from each format's binary fields with the same XMSS signatures (field sizes in Online Resource 1): 2.5, 3.7, 8.9, and 12.4 KiB for the Merkle log, SDLS with CFDP, signed syslog, and NEITH-ACE, or 1.5, 2.2, 5.3, and 7.3% of the telemetry volume. These are format counts; the reference implementation's length-prefixed encoding measures 17.7 KiB, or 10.4%. Manifest signatures are the largest part and shrink in proportion to a longer manifest period Δ_m; the 32-byte digest per chunk does not.

Finally, the capture adapter was built as an active component of F′, which has flown on CubeSats such as ASTERIA [20], and deployed on the Pi 4, one instance per sensor with its own keys and state file. It tags with prefix-keyed Ascon-CXOF128, digests with SHAKE128, and seals a chained manifest every Δ_m one-second ticks; on fixed test vectors its tags, digests, genesis value and manifest signing message match the Python reference byte for byte, on the Pi 4 as on an x86 host, and an XMSS signature it produces verifies. Two instances replayed recorded GNSS and LiDAR streams, eight GNSS lines and two 512-byte LiDAR blocks per tick. The component's log shows 656 GNSS chunks in nine manifests ending in a final one, and 280 LiDAR chunks in 14 manifests before the run was ended manually, with contiguous numbers and periodic manifests every 10.00 s at the median (9.88–10.17 s). Sealing, which covers signing and the durable write of the XMSS state, took a 54.8 ms median and 220 ms at worst, under 0.6% and about 2.2% of the period.

7 Related Work

Forward-integrity audit logs [21, 9] protect entries written before a compromise but not the end of the log, and aggregate schemes [22, 23] add resistance to truncation of those entries; verifiable excerpts [24] extend completeness checks to parts of a log. Nitro brings this symmetric approach to high-throughput kernel auditing and reports that earlier tamper-evident loggers drop most entries under load [19, 25]. All of them trust the logger to receive what it should; none can show that it left out entries it never received, the loss NEITH-ACE's manifests expose, and their symmetric keys let only key holders verify. History trees [18] and Certificate Transparency [2] let auditors check an untrusted log's membership and consistency; NEITH-ACE borrows their signed head, but entries never submitted to such a log lie outside its scope.

Closest in purpose are source-signed flight logs. LiveBox records flight data on a ledger [26], and in DASLog the uncrewed aerial vehicle (UAV) and other sources sign hash chains of their records, closed by a final message and anchored on a permissioned blockchain, which exposes an operator that drops or truncates records [6]. NEITH-ACE shares the idea of source signatures but attributes each loss to the last authenticated boundary it crossed, repairs recoverable gaps, needs no validator set, and uses post-quantum signatures. Secure failure localization [27] locates the network hop that drops packets, a related goal for traffic rather than stored records. Of the standardized hash-based signatures, XMSS [11, 12] and LMS/HSS [28] are stateful, with signatures of a few kilobytes but no security if a one-time key signs twice [29, 30]; stateless SLH-DSA [31] needs at least 7856-byte signatures and signs far more slowly. NEITH-ACE signs once per window or checkpoint in every role, so it uses XMSS and inherits its state discipline; LMS/HSS would serve equally.

8 Conclusion

NEITH-ACE turns authenticated telemetry into an authenticated account of what a record holds and what it lacks. Beyond what source-signed schemes already give, it attributes every missing chunk to the last authenticated boundary it crossed and repairs recoverable gaps, with hash-based signatures, at about 1.4 times the evidence of signed syslog and five times that of a Merkle log, counted the same way. The whole pipeline sustained 1598 chunks/s on a Raspberry Pi 4, about 120 times the live load.

The guarantee has limits. It is relative to what the adapters declare, so it says nothing about the physical truth of a reading, and it reaches the end of a stream only as far as the last declaration. It assumes the adapters' keys are kept from the integrity service and that the verifier receives its keys before launch; the testbed ran adapters and service as separate processes, not in separate hardware partitions. Under a compromised service only the declared record and, while the store does not collude, the store's receipts bind. The verifier was checked by the proofs and the 44 injector scenarios, not formally verified. On the measurement side, the injector simulates time, the live run predates two later protocol refinements that do not affect it (Online Resource 1), and the live pipeline covered one

epoch on one edge platform and one audit platform. Inside F′, the capture adapter read recorded sensor streams rather than a live serial-port (UART) driver, and its frame output was stubbed, so its chunks and manifests were logged but not transmitted or verified; the end-to-end results come from the host-process adapters. An F′ build of the store was compiled but never deployed. F′ supplies scheduling and the component framework; the security rests on the adapter's key and the verifier's checks. The $h = 20$ key generation was not measured.

For an investigator reconstructing a spacecraft anomaly, the result is a record that states what is missing and the last point at which each missing chunk was still accounted for. Attesting the service would narrow the compromised case, and a handoff witness would split HANDOFFABSENT between service and store.

Acknowledgements The author thanks the Reconfigurable Space Computing Laboratory at California State Polytechnic University, Pomona, for supporting the edge platform used in this work. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

Supplementary information Online Resource 1 contains the notation, the complete verifier algorithm, domain tags, full proofs, the attack coverage table, key lifetimes, the evidence-cost model, and reproduction details.

Statements and Declarations

Funding This research was funded by the U.S. Navy Naval Engineering Education Consortium (NEEC; Grant N00174-23-1-0002) and the Office of Naval Research (ONR) Summer Faculty Research Program (SFRP). It was also supported in part by the U.S. Army Research Office (U.S. Department of Defense) under award W911NF-24-1-0265 and by the Air Force Research Laboratory (AFRL) under agreement FA8650-24-2-2403. AMD provided development boards in kind. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon.

Competing interests The author declares no competing interests.

Ethics approval and consent Not applicable: no human participants, animals, or personal data.

Consent for publication Not applicable.

Author contributions The author carried out all of the work and wrote the article.

Use of AI tools Generative AI tools assisted with manuscript drafting and revision, portions of the software implementation, and consistency checks across the manuscript, supplementary material, and source code. The author conceived the research idea, designed the architecture and protocol, conducted the experiments, and retained responsibility for the research design, scientific interpretation, validation of results, and final manuscript content.

Data and code availability The reference implementation, F′ components, attack injector, benchmark scripts, and all measurement logs (directory `results`) are available under the MIT licence at `https://github.com/mealycpp/neith-ace`.

Materials availability Not applicable.

Extended version This article extends the IEEE SATC 2026 paper NEITH [1]; only its window and root construction (Section 2) and sensors, now on a Pi 4 instead of a Pi 5, carry over.

References

[1] Mohamed El-Hadedy. NEITH: Networked evidence integrity for telemetry hashing. In *2026 IEEE 2nd International Conference on Secure IoT, Assured and Trusted Computing (SATC)*, pages 1–5, 2026.

[2] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. RFC 6962, 2013.

[3] CCSDS. Space data link security protocol. Recommended Standard (Blue Book) CCSDS 355.0-B-2, Consultative Committee for Space Data Systems, 2022.

[4] CCSDS. CCSDS file delivery protocol (CFDP). Recommended Standard (Blue Book) CCSDS 727.0-B-5, Consultative Committee for Space Data Systems, 2020.

[5] John Kelsey, Jon Callas, and Alexander Clemm. Signed syslog messages. RFC 5848, 2010.

[6] Roozbeh Sarenche, Seyed Farhad Aghili, Takahito Yoshizawa, and Dave Singelée. DASLog: Decentralized auditable secure logging for UAV ecosystems. *IEEE Internet of Things Journal*, 10(23):20264–20284, 2023.

[7] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. FIPS 202, NIST, 2015.

[8] Meltem Sönmez Turan, Kerry A. McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. Ascon-based lightweight cryptography standards for constrained devices. Special Publication 800-232, NIST, 2025.

[9] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, 1999.

[10] Mark L. Psiaki and Todd E. Humphreys. GNSS spoofing and detection. *Proceedings of the IEEE*, 104(6):1258–1270, 2016.

[11] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle signature scheme. RFC 8391, 2018.

[12] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for stateful hash-based signature schemes. Special Publication 800-208, NIST, 2020.

[13] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography – PKC 2016*, volume 9614 of *LNCS*, pages 387–416, Berlin, Heidelberg, 2016. Springer.

[14] John Kelsey, Shu-jen Chang, and Ray Perlner. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. Special Publication 800-185, NIST, 2016.

[15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In *Fast Software Encryption – FSE 2015*, volume 9054 of *LNCS*, pages 364–384, Berlin, Heidelberg, 2015. Springer.

[16] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS – a practical forward secure signature scheme based on minimal security assumptions. In *Post-Quantum Cryptography (PQCrypto)*, volume 7071 of *LNCS*, pages 117–129, Berlin, Heidelberg, 2011. Springer.

[17] Mohamed El-Hadedy. KHEPRI: Verifiable perception windows with keepalive and hot-swap recovery on edge microclusters. In *2026 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pages 2161–2166, 2026.

[18] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *Proc. 18th USENIX Security Symposium*, pages 317–334, 2009.

[19] Rui Zhao, Muhammad Shoaib, Viet Tung Hoang, and Wajih Ul Hassan. Rethinking tamper-evident logging: A high-performance, co-designed auditing system. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2025. arXiv:2509.03821.

[20] Robert Bocchino, Timothy Canham, Garth Watney, Leonard Reder, and Jeffrey Levison. F prime: An open-source framework for small-scale flight software systems. In *Proc. 32nd Annual AIAA/USU Conference on Small Satellites*, 2018. SSC18-XII-04.

[21] Mihir Bellare and Bennet Yee. Forward integrity for secure audit logs. Technical report, University of California, San Diego, Department of Computer Science and Engineering, 1997.

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

[23] Attila A. Yavuz, Peng Ning, and Michael K. Reiter. BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. *ACM Transactions on Information and System Security*, 15(2):9, 2012.

[24] Gunnar Hartung. Secure audit logs with verifiable excerpts. In *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *LNCS*, pages 183–199, Cham, 2016. Springer.

[25] Riccardo Paccagnella, Pubali Datta, Wajih Ul Hassan, Adam Bates, Christopher W. Fletcher, Andrew Miller, and Dave Tian. Logging to the danger zone: Race condition attacks and defenses on system audit frameworks. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 1551–1574, 2020.

[26] Yijun Yu, Danny Barthaud, Blaine A. Price, Arosha K. Bandara, Andrea Zisman, and Bashar Nuseibeh. LiveBox: A self-adaptive forensic-ready service for drones. *IEEE Access*, 7:148401–148412, 2019.

[27] Boaz Barak, Sharon Goldberg, and David Xiao. Protocols and lower bounds for failure localization in the Internet. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 341–360, Berlin, Heidelberg, 2008. Springer.

[28] David McGrew, Michael Curcio, and Scott Fluhrer. Leighton-Micali hash-based signatures. RFC 8554, 2019.

[29] Leon Groot Bruinderink and Andreas Hülsing. "Oops, I did it again" – security of one-time signatures under two-message attacks. In *Selected Areas in Cryptography – SAC 2017*, volume 10719 of *LNCS*, pages 299–322, Cham, 2018. Springer.

[30] David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. State management for hash-based signatures. In *Security Standardisation Research (SSR 2016)*, volume 10074 of *LNCS*, pages 244–260, Cham, 2016. Springer.

[31] National Institute of Standards and Technology. Stateless hash-based digital signature standard. FIPS 205, NIST, 2024.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- ESM1.pdf