Journal of Hardware and Systems Security

Online Resource 1

Supplementary material for "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

References to the main article's equations, algorithm lines, tables, and sections say so explicitly; theorems keep the main article's numbers, and tables here are numbered S1, S2, and so on. The adversary $\mathcal{A}$ is as in Section 2 and the advantages $\mathrm{Adv}^{\mathrm{cr}}_H$ (finding a collision in H) and $\mathrm{Adv}^{\mathrm{euf}}_X$ (forging against X under chosen-message attack) as in Section 4 of the main article; Sign and Verify are the signing and verification algorithms of the scheme named by the key.

Abbreviations. Ascon: the NIST lightweight-cryptography family, with Ascon-XOF128 and Ascon-CXOF128 its extendable-output functions (XOF, and CXOF with a customization string); BDS: the Buchmann–Dahmen–Szydlo tree traversal used to compute XMSS signatures; CFDP: CCSDS File Delivery Protocol; CCSDS: Consultative Committee for Space Data Systems; F′: NASA JPL's flight-software framework; GNSS: global navigation satellite system; KMAC: Keccak message authentication code; LiDAR: light detection and ranging; LSB: least significant bit; MAC: message authentication code; NIST: U.S. National Institute of Standards and Technology; NMEA: the NMEA 0183 text format of GNSS receivers; PDU: protocol data unit; RFC: IETF Request for Comments; SDLS: CCSDS Space Data Link Security; SHAKE: the SHA-3 extendable-output functions [1]; SP: NIST Special Publication; SPI: security parameter index; TCP: Transmission Control Protocol; TM: telemetry transfer frame; UTF-8: the Unicode byte encoding; WOTS+: the Winternitz one-time signature inside XMSS; XMSS: eXtended Merkle Signature Scheme.

1 Notation

Table S1: Symbols, domains, and units.

Symbol	Meaning
Time and schedule	
$\Delta > 0$, $0 \le \gamma < \Delta$	window length and grace period (s)
τ_0, τ, T_t, E_t	epoch start, ingestion time, close of window t, emission time of R_t (s)
τ_a	audit time at which the verifier runs (s)
$\Delta_h > 0$, $\kappa > 0$	heartbeat period (s) and staleness multiplier

Symbol	Meaning
$k \geq 1$	checkpoint period (windows)
θ, θ_{st}	publication allowance after $T_t + \gamma$ and receipt deadline after T_t (s)
d_{pub}, ϵ	bound on honest publication delay and clock skew between service and verifier (s)
Δ_m, B	manifest period (s; rate-group ticks in F′) and resend-buffer capacity (chunks)
Indices and parties	
e, $t \geq 0$, $s \geq 0$	epoch, window, sequence number
$i \in \{1, \ldots, m\}$, A_i	sensor and its capture adapter, encoded as the integer $i-1$; m is the number of sensors
$v \geq 1$, u_i	manifest index and strictly increasing heartbeat counter
$\hat{s}_i$	highest sequence number sent by A_i (-1 before the first chunk)
$\hat{s}^*$, t^*	largest sequence number declared in a verified history (Theorem 3) and checkpoint index (Theorems 2 and 3)
$\mathcal{E}$	sequence numbers declared in A_i's manifest chain up to the audit scope (Theorem 4)
Keys and primitives	
H, MAC, enc	32-byte collision-resistant hash; KMAC128 or keyed Ascon-CXOF128; injective, prefix-free, length-prefixed encoding
Σ, Σ^A, Σ_{st}	signature schemes for checkpoints, manifests, and receipts (all XMSS-SHAKE256)
K_i	MAC key shared by A_i and the integrity service
sk, pk	checkpoint key pair of the integrity service
sk_i^A, pk_i^A	manifest key pair of A_i; the service never holds sk_i^A
sk_{st}, pk_{st}	receipt key pair of the store
cfg	configuration (parameters and declaration mode) fixed in the genesis R_{-1}
h	XMSS tree height (distinct from the header h_t)
Adapter evidence	
$c_{i,s}$, $\alpha_{i,s}$	chunk payload and its MAC tag
Π	manifest open list of pairs $(s, H(c_{i,s}))$ since the previous manifest (P remains the worker count)
$\mu_{i,v}$, *fin*	manifest-body hash ($\mu_{i,0} = H(\texttt{0x09} \| \mathsf{enc}(e,i))$) and final flag
Records	
$\rho_{i,t}$	record of sensor i in window t, $(i, a, b, \mathcal{G}^{\mathrm{miss}}, \mathcal{G}^{\mathrm{auth}}, \mathcal{L}, \mathcal{Q}, r, \hat{s}, u, \phi, \mathcal{M})$
$a_{i,t}, b_{i,t}$	interval of sequence numbers window t accounts for
$\mathcal{G}^{\mathrm{miss}}$, $\mathcal{G}^{\mathrm{auth}}$, $\mathcal{L}$	missing, authentication-failed, and late-arrival sets (calligraphic $\mathcal{L}$ is distinct from chunk size L)
$\mathcal{Q}_{i,t}$, $r_{i,t}$	receipt-acknowledged pairs (t', s) and receipt digest
$\phi_{i,t}$, $\mathcal{M}_{i,t}$	staleness flag and manifests arriving in window t
Tree, header, checkpoint, receipt	
N_t, n_t, $N_{\max}$	leaves, arrivals, and leaf bound per window
x_L, x_R	left and right inputs of a tree node $H(\texttt{0x01} \| x_L \| x_R)$
Λ_t, D_t, R_t	tree root, record digest, window root
h_t	header (t, N_t, Λ_t, D_t): 80 bytes raw, 96 bytes as a standalone enc

Symbol	Meaning
β_t, q_t	$\lceil N_t/8 \rceil$-byte receipt bitmap (bit j = leaf position j, LSB first) and store receipt
σ_t, Ω	checkpoint signature and key lifetime (s)
Costs	
P, L, λ, $\lambda_{\max}$	workers, chunk size (bytes), aggregate chunk rate and highest sustainable rate (chunks/s)
$c_{\mathsf{MAC}}, c_H, c_{\mathrm{n}}, c_\rho, c_{\mathrm{root}}, c_{\mathrm{sig}}$	per-operation costs (s); c_{n} is one 65-byte node hash, and c_{sig} includes durable state writes
W_t, I_t	work to close window t (s); $I_t = 1$ in checkpoint windows, 0 otherwise
C, S	per-chunk parallel cost $c_{\mathsf{MAC}} + c_H + c_{\mathrm{n}}$ and serial remainder of Eq. (4) of the main article at $N_{\max}$ (s)

2 Verifier

Algorithm S1 is the verifier of the reference code (class `Verifier` in `neith_ace.py`). The verifier holds cfg and every public key from provisioning; the genesis root it is handed must equal the one these inputs determine. The scope of an audit is the checkpointed prefix of the history.

Algorithm S1 NEITH-ACE verifier (it may continue across pages).

Require: provisioned cfg (with e, k, declaration mode), pk, $\langle pk_i^A \rangle_i$, pk_{st}; handed genesis $\tilde{R}_{-1}$; w published windows $(h_j, \langle \rho_{i,j} \rangle_i, \tilde{R}_j, \sigma_j)$ at positions $j = 0, \ldots, w-1$; receipts $\{(q_t, \beta_t)\}$; store access (optional); audit time τ_a

Genesis and chain

if $\tilde{R}_{-1} \neq H(\mathtt{0x06} \| \mathsf{enc}(e, \mathsf{cfg}, pk, \langle pk_i^A \rangle, pk_{\mathrm{st}}))$ **then return** INVALID
if signed declarations are configured and $\langle pk_i^A \rangle$ is empty **then return** INVALID
$R_{-1} \leftarrow \tilde{R}_{-1}$; $c^* \leftarrow -1$; $t_{\mathrm{end}} \leftarrow \infty$ ▷ last checkpointed window; end of schedule
for $j = 0, \ldots, w-1$, with $h_j = (t, N_t, \Lambda_t, D_t)$ **do**
  if $t \neq j$, or $D_t \neq H(\mathtt{0x02} \| \mathsf{enc}(\rho_{1,t}, \ldots, \rho_{m,t}))$ **then return** INVALID
  $R_t \leftarrow H(\mathtt{0x05} \| \mathsf{enc}(e, t, N_t) \| \Lambda_t \| D_t \| R_{t-1})$; **if** $R_t \neq \tilde{R}_t$ **return** INVALID
  if σ_t absent, $t \equiv k-1 \pmod k$, and $T_t + \gamma + \theta \leq \tau_a$ **then return** WITHHELD ▷ before its deadline it is pending
  if σ_t present **then**
    if $\mathsf{Verify}(pk, H(\mathtt{0x07} \| \mathsf{enc}(e, t, 0) \| R_t), \sigma_t)$ **then** $c^* \leftarrow t$
    else if $\mathsf{Verify}(pk, H(\mathtt{0x07} \| \mathsf{enc}(e, t, 1) \| R_t), \sigma_t)$ and $j = w-1$ **then** $c^* \leftarrow t$; $t_{\mathrm{end}} \leftarrow t$ ▷ shutdown
    else return INVALID
if $c^* < w-1$ **then** ▷ unsigned tail
  $t' \leftarrow c^* + k$ if $c^* \geq 0$, else $k-1$
  if $T_{t'} + \gamma + \theta \leq \tau_a$ **then return** WITHHELD
  mark windows $c^*+1, \ldots, w-1$ pending and exclude them from the scope
if some $t \equiv k-1 \pmod k$ with $c^* < t \leq \min(t(\tau_a), t_{\mathrm{end}})$, $t(\cdot)$ the window map of Eq. (1) of the main article, has $T_t + \gamma + \theta \leq \tau_a$ **then return** WITHHELD

Records, windows in scope in order; per sensor $\nu_i \leftarrow 0$, open missing set O_i, auth-failed set U_i, repair list $\mathcal{R}$, *stale* $\leftarrow$ false, *unconf* $\leftarrow$ false, *pend* $\leftarrow$ false

for each record $\rho_{i,t} = (i, a, b, \mathcal{G}^{\mathrm{miss}}, \mathcal{G}^{\mathrm{auth}}, \mathcal{L}, \ldots, \phi, \mathcal{M})$ **do**
  if $a \neq \nu_i$ **then return** INVALID
  for $s \in \mathcal{L}$ **do if** $s \notin O_i \cup U_i$ **return** INVALID; append (i, s) to $\mathcal{R}$; remove s from O_i, U_i
  for $s \in \mathcal{G}^{\mathrm{auth}}$ **do if** ($s < a$ and $s \notin O_i$) or $s > b$ **return** INVALID; **if** $s < a$ remove s from O_i
  for $s \in \mathcal{G}^{\mathrm{miss}}$ **do if** $s \notin [a, b]$ or $s \in \mathcal{G}^{\mathrm{auth}}$ or $s \in O_i$ **return** INVALID
  $O_i \leftarrow O_i \cup \mathcal{G}^{\mathrm{miss}}$; $U_i \leftarrow U_i \cup \mathcal{G}^{\mathrm{auth}}$; $\nu_i \leftarrow b+1$; *stale* $\leftarrow$ *stale* $\vee \phi$

23: $\mathcal{X} \leftarrow \{(i, s, \textsc{IngestAbsent}) : s \in O_i\} \cup \{(i, s, \textsc{IngestAuthFail}) : s \in U_i\}$ ▷ absence set

Manifests (signed mode), in window order over all sensors, each sensor's chain starting from $(0, \mu_{i,0})$

24: **for** each manifest $(v, \hat{s}, \mu', \mathit{fin}, \Pi, \varsigma)$ of each A_i, with signature ς and v_{prev} the index of A_i's previous manifest (0 before the first) **do**

25: **if** $v \neq v_{\text{prev}} + 1$ **then return** Withheld ▷ a manifest was skipped

26: **if** $\mu' \neq \mu_{i,v-1}$ **then return** Invalid ▷ wrong chain link

27: $\mu_{i,v} \leftarrow H(\texttt{0x0a} \| \mathsf{enc}(\texttt{"mf"}, e, i, v, \hat{s}, \mu', \mathit{fin}, \Pi))$

28: **if** a *fin* manifest of A_i was already seen, or $\neg\mathsf{Verify}(pk_i^A, \mu_{i,v}, \varsigma)$ **then return** Invalid

29: **if** some s in Π is already declared **return** Invalid; declare $d_{i,s} \leftarrow H(c_{i,s})$ from Π; frontier $f_i \leftarrow \hat{s}$

30: for each i: **if** $f_i \geq 0$ and the declared numbers of A_i are not exactly $\{0, \ldots, f_i\}$ **return** Invalid ▷ $f_i = -1$ if none

31: for each i: **if** $\nu_i - 1 > f_i$: *unconf* $\leftarrow$ true; **if** $\nu_i - 1 < f_i$: add $(i, s, \textsc{IngestAbsent})$ to $\mathcal{X}$ for $\nu_i \leq s \leq f_i$

Identities

32: for each window t in scope, its leaf identities: $\mathcal{L}$ and $[a, b] \setminus (\mathcal{G}^{\text{miss}} \cup \mathcal{G}^{\text{auth}})$ of each record, in (i, s) order; **if** their number is not N_t **return** Invalid

33: **if** any identity (i, s) is committed twice in the scope **then return** Invalid

34: **for** each committed (i, s) with no $d_{i,s}$ (signed mode) **do**

35: **if** $s \leq f_i$ or A_i has a *fin* manifest **return** Invalid; **else** *unconf* $\leftarrow$ true

Custody, windows t in scope, positions $p < N_t$ with identity (i, s)

36: $\mathcal{Z}_t \leftarrow$ set bits of β_t if $|\beta_t| = \lceil N_t/8 \rceil$ bytes, no bit beyond position $N_t - 1$ is set, and $\mathsf{Verify}(pk_{\text{st}}, \mathsf{enc}(\texttt{"rcpt"}, e, t, N_t) \| \Lambda_t \| \beta_t, q_t)$, else $\emptyset$

37: *opened* $\leftarrow$ the store returns (i, s, c, π) with $|\pi| = \lceil \log_2 N_t \rceil$ and π leads from $H(\texttt{0x00} \| \mathsf{enc}(e, t, i, s) \| c)$ at p to Λ_t; a returned opening that fails this check is recorded as altered

38: **if** *opened* and $d_{i,s}$ exists and $H(c) \neq d_{i,s}$ **return** Invalid

39: **if** $p \in \mathcal{Z}_t$ **then if** a store is available and not *opened*: add $(i, s, \textsc{CustodyLoss})$ to $\mathcal{X}$

40: **else if** $\tau_a \geq T_t + \theta_{\text{st}}$ **then** add $(i, s, \textsc{HandoffAbsent})$ to $\mathcal{X}$

41: **else** *pend* $\leftarrow$ true ▷ receipt not yet due, opened or not

Verdict

42: *stale* $\leftarrow$ *stale* $\vee$ (*unconf* and no tail is pending) ▷ a pending tail holds the missing manifests

43: **if** signed mode, some A_i has no *fin* manifest in scope or a tail is pending, $\mathcal{X} = \emptyset$, and $\neg$*stale* **then return** Prefix

44: **if** *stale* **then return** Stale

45: **if** $\mathcal{X} \neq \emptyset$ **then return** Accounted($\mathcal{X}$)

46: **if** *pend* **then return** Prefix ▷ a receipt not yet due: never Complete or Repaired early

47: **if** a tail is pending or no store is available **then return** Prefix

48: **return** Repaired if $\mathcal{R} \neq \emptyset$, else Complete

---

Algorithm S1 is the reference verifier for an audit at time τ_a with the service assumed reachable throughout; the code can also take the contact windows of a ground station, in which case a deadline counts as passed only if a contact followed it. A scheduled checkpoint that has not appeared is pending until its deadline $T_t + \gamma + \theta$ and Withheld only afterwards, so an honest service audited between closing a window and publishing its checkpoint is not accused. The checkpoint test in line 8 applies to every window that carries a signature. The shutdown checkpoint of line 10 of Algorithm 1 in the main article signs *end* $= 1$, which is valid only on the last published window; it ends the schedule, so line 16 expects no later checkpoint even when shutdown falls on a scheduled checkpoint window, and an honest history audited long after shutdown is not reported as withheld. The injector tests shutdown on a scheduled checkpoint window with an on-time audit, a late audit, and a withheld shutdown checkpoint, and an end checkpoint forged in mid-history (Table S3).

The service commits manifests in chain order and re-requests a missing one, holding the later ones until it arrives, so an honest record never skips an index; a skipped index is therefore suppression (Withheld), and a present manifest whose predecessor hash is wrong is Invalid. If the missing manifest never arrives, the later ones stay held, the declaration frontier stops, and the chunks committed beyond it make the history Stale. Committed chunks beyond the last manifest in scope are unconfirmed (*unconf*); they make the history Stale unless the scope was cut short only by a checkpoint not yet due, in which case their manifests lie in the pending tail and the result is Prefix.

Line 37 opens every committed position $p = 0, \ldots, N_t - 1$ of every window in scope, receipted or not, so a position the store does not mention is still checked. An opening counts only if it verifies; one the store returns but

that fails verification is reported separately as altered and, like an unavailable one, attributed to custody (or to the handoff if never receipted), never INVALID: the store is untrusted, and its misconduct must not let it invalidate the history the service committed. The same reasoning makes a malformed receipt, with the wrong length or with bits set beyond position $N_t - 1$, acknowledge nothing rather than make the history INVALID. A receipted position the store cannot produce is CUSTODYLOSS; a position never receipted by $T_t + \theta_{\mathrm{st}}$ is HANDOFFABSENT even if the store can still produce it, because the store never accepted custody of it; before that deadline the position is pending even if the store can produce it, so the history is at most PREFIX and never COMPLETE or REPAIRED early. PREFIX never claims completeness: if a compromised service suppressed a sensor's entire manifest chain together with its records, the verdict is PREFIX or STALE, and an investigator who knows the mission has ended reads a missing final manifest as suppression.

Four record fields do not enter the verdict. The heartbeat counter u only lets the service reject replayed heartbeats, and the recorded $\hat{s}$ is the service's own claim, used by Theorem 3 only when the service is honest. $\mathcal{Q}_{i,t}$ and $r_{i,t}$ record inside the committed history which receipts the service had received by window t; the verifier checks the store's signed receipts directly.

3 Domain separation

Table S2: Domain-separation tags.

Tag	Role	Tag	Role
`0x00`	leaf	`0x06`	genesis root
`0x01`	internal node	`0x07`	checkpoint message ($\mathsf{enc}(e, t, \mathit{end}) \| R_t$)
`0x02`	sensor-record digest	`0x08`	receipt digest
`0x03`	padding node	`0x09`	manifest-chain genesis
`0x04`	empty tree	`0x0a`	manifest body
`0x05`	window root		

In a record, $\mathcal{Q}_{i,t}$ lists the accepted pairs (t', s) and $r_{i,t} = H(\texttt{0x08} \,\|\, \mathsf{enc}(y_1, y_2, \dots))$, each $y = \mathsf{enc}(t', q_{t'}, \beta_{t'})$ a byte string, digests the receipts that arrived in window t (byte layout in the reproduction section below). The payload digest $H(c)$ inside a manifest is the one untagged hash: it is only ever compared with another payload digest and enters every protocol hash as a length-prefixed field.

4 Full proofs

Window binding. An *opening* of a window root R_t consists of the previous root R_{t-1}, the header $h_t = (t, N_t, \Lambda_t, D_t)$ with epoch e, the encoded record list behind D_t, and the ordered list of leaf preimages (e, t, i, s, c) behind Λ_t, which has exactly N_t entries. Two openings conflict if they differ in any component. Because `enc` is injective and prefix-free and Λ_t, D_t, R_{t-1} are fixed-length, the input of every tagged call parses in exactly one way.

Lemma S1 (Path binding)**.** *Let Λ_t be the root of the committed tree over N_t leaves. If a store returns, for position p, a leaf preimage (i, s, c) and a sibling path π with $|\pi| = \lceil \log_2 N_t \rceil$ that leads to Λ_t, then the preimage equals the committed leaf preimage at p, unless H collides across calls tagged `0x00`, `0x01`, and `0x03`.*

Proof. Padding pairs every odd node at its level, so every leaf of the committed tree lies at depth exactly $\lceil \log_2 N_t \rceil$, and p fixes the left or right step at each level. The returned path therefore has the same length and step sequence as the committed path to p. Recompute both from the root down. At each level both nodes equal the same value; if their inputs (x_L, x_R) differ, the two `0x01` calls collide, and otherwise the child on the path to p agrees and the walk continues. At the bottom the two leaf values agree, so differing preimages collide in the `0x00` call. A path of the wrong length would compare a leaf with an internal or padding node; it is rejected by the length check, and an equal value across the distinct tags `0x00`, `0x01`, `0x03` would itself be a collision. □

Theorem 1 (Window binding)**.** *For any adversary $\mathcal{A}$ that outputs two conflicting openings of the same value R_t, there exists an adversary $\mathcal{B}$ with comparable running time such that* $\Pr[\mathcal{A} \text{ succeeds}] \le \mathrm{Adv}^{\mathrm{cr}}_H(\mathcal{B})$.

Proof. $\mathcal{B}$ runs $\mathcal{A}$ and recomputes both openings top-down. If the two inputs to the `0x05` call in line 8 of Algorithm 1 in the main article differ, they form a collision. Otherwise (e, t, N_t), Λ_t, D_t, and R_{t-1} agree. D_t fixes the records: equal D_t with different record lists yields a collision in the `0x02` call. N_t fixes the tree shape, because the shape of a padded binary tree is a function of its leaf count alone, and both leaf lists have N_t entries. Walking both trees from the root, the first node at which the children differ supplies a collision in an `0x01` call, and if every internal node agrees, two differing leaves supply a collision in an `0x00` call, as in Lemma S1. Padding nodes cannot be confused with internal nodes or leaves because their tag `0x03` differs from both, and the empty-tree constant `0x04` is reachable only when $N_t = 0$. Hence every success of $\mathcal{A}$ yields a collision for $\mathcal{B}$. □

Inclusion soundness, exact leaf count, and immutability of every gap set and late-arrival list follow directly, because each is a component of the opening; Lemma S1 carries the same guarantee to a single position opened by the store. The conference version bound only the leaf multiset; it left N_t and the window index outside the hash, so two windows with differently padded trees could share a tree root. Theorem 1 closes that gap.

History integrity. Let t^* be a checkpoint index and σ_{t^*} a signature that verifies under pk on

$$H(\texttt{0x07} \| \mathsf{enc}(e, t^*, end) \| R_{t^*}),$$

with $end \in \{0, 1\}$ and $end = 1$ only on the last published window.

Theorem 2 (History integrity)**.** *Assume the checkpoint signer is honest, its XMSS state is never rolled back, and no epoch number is reused under its key. For any adversary $\mathcal{A}$ that outputs a valid checkpoint for epoch e and index t^* together with a sequence of openings from R_{-1} to R_{t^*} that differs from the sequence the integrity service committed, there exist adversaries $\mathcal{B}_1, \mathcal{B}_2$ with comparable running time such that* $\Pr[\mathcal{A} \text{ succeeds}] \le \mathrm{Adv}^{\mathrm{euf}}_{\Sigma}(\mathcal{B}_1) + \mathrm{Adv}^{\mathrm{cr}}_{H}(\mathcal{B}_2)$.

Proof. $\mathcal{B}_1$ simulates the honest service with its signing oracle. If the checkpoint message was never queried, $\mathcal{B}_1$ outputs it as a forgery against Σ; the reduction is sound because the state is never rolled back, so no one-time index is used twice. Otherwise the service signed $H(\texttt{0x07} \| \mathsf{enc}(e', t', end') \| R')$ for some (e', t', end') and root R'. Epochs are unique under the key and the service signs each window at most once, with one value of end, so if $(e', t', end', R') \neq (e, t^*, end, R_{t^*})$ the two `0x07` inputs collide, a collision that $\mathcal{B}_2$, running the same simulation, outputs; hence the presented chain ends in the committed root R_{t^*}. It starts in the same genesis, because the verifier computes R_{-1} from its own provisioned inputs (line 1 of Algorithm S1), so the genesis needs no collision term. The verifier also checks that the window index equals the position, so both sequences cover windows $0, \ldots, t^*$. $\mathcal{B}_2$ holds both sequences and compares them from t^* downward; the reduction finds the first differing window directly, without guessing. At that window both sequences open the same root, since every later window agreed and therefore fixed the same predecessor, and Theorem 1 turns the two conflicting openings into a collision. □

The theorem covers deletion, insertion, reordering, and substitution of whole windows, the attacks that the conference version [2] could detect only through schedule timeouts at an archival verifier. For the XMSS instantiation the bound holds only while the one-time index never repeats: a rolled-back state signs two messages under one WOTS+ key, which exposes enough chain values to forge further signatures [3], and $\mathcal{B}_1$ then no longer reduces to unforgeability. A compromised service holding sk can sign a different chain, which is why Theorem 4 does not depend on this theorem.

Declared completeness. This statement belongs to the honest-service case. Fix an honest adapter A_i, an integrity service that computes records as specified, and a committed history that verifies through checkpoint t^*. Let $\hat{s}^*$ be the largest sequence number declared in any heartbeat recorded in that history, or carried by any validly tagged chunk recorded there. The MAC game gives the adversary tagging and verification queries: every chunk or heartbeat it injects on the link is a verification query answered by the service's tag check.

Theorem 3 (Declared completeness)**.** *Assume adapter A_i and the integrity service are honest, K_i is secret, and the hypotheses of Theorem 2 hold. Except with probability* $\mathrm{Adv}^{\mathrm{euf}}_{\mathsf{MAC}} + \mathrm{Adv}^{\mathrm{euf}}_{\Sigma} + \mathrm{Adv}^{\mathrm{cr}}_{H}$*, every sequence number $s \le \hat{s}^*$ ends, in the reconciled history of sensor i (each later repair applied to the gap it closes), either as exactly one leaf carrying the payload that A_i tagged or as an unresolved gap, never both.*

Proof. The record intervals of sensor i tile $[0, b_{i,t}]$ by construction, and $b_{i,t} \ge \hat{s}^*$ at the window that recorded $\hat{s}^*$. The service classifies every interval member as a leaf or a gap when it closes the window, and a late arrival adds a leaf only for a number already in an earlier gap set, so each number occurs once as a leaf or once as an unresolved gap. A leaf whose payload differs from the payload A_i tagged requires a valid tag on a message A_i never authenticated, which is a MAC forgery. A declared value $\hat{s}^*$ larger than anything A_i sent likewise requires a forged heartbeat. Theorem 2 transfers the statement from the service's view to the verifier's view at the cost of its own bound. □

No custody scheme can detect the loss of chunks produced after the last declaration: an adversary who drops the final chunks and every later heartbeat leaves no declaration to contradict. NEITH-ACE limits that interval instead of hiding it. The staleness flag fires after $\kappa\Delta_h$, so the verifier learns that the sensor's account ends at $\hat{s}^*$ and that nothing after it is covered, to within $\kappa\Delta_h$ plus one window.

Completeness under a compromised service. Theorem 3 assumes the service computes records as specified; it is the right statement when the service is trusted, as in the conference version. Once the service is not trusted, it holds every admission key K_i and its own checkpoint key, so a MAC heartbeat is no longer evidence to a third party: the service can forge heartbeats, fabricate leaves, drop chunks, and sign a chain that calls the result complete. The defense is the signed manifest of Section 3 of the main article. Each adapter signs, under its own key sk_i^A that the service never holds, a chained manifest declaring the digest of every captured chunk, and a FINAL manifest closes the epoch. The verifier derives the expected identifier set from these manifests, not from the service's records.

Theorem 4 (No false completeness)**.** *Let A_i be an honest adapter whose XMSS state never repeats an index, with secret signing key sk_i^A. Let the verifier hold* `cfg` *and pk_i^A from provisioning and have access to the store. Let the integrity service be arbitrarily malicious and possibly colluding with the store. Let $\mathcal{E}$ be the set of sequence numbers declared in A_i's manifest chain, closed by a* FINAL *manifest, up to the scope of the audit. Except with probability $\mathrm{Adv}^{\mathrm{euf}}_{\Sigma^A} + \mathrm{Adv}^{\mathrm{cr}}_{H}$, the verifier of Algorithm S1 returns* COMPLETE *or* REPAIRED *for sensor i only if a final manifest of A_i lies inside the checkpointed scope, the committed leaves of sensor i are exactly $\mathcal{E}$, each once, and for every $s \in \mathcal{E}$ the store returns a payload whose digest equals $H(c_{i,s})$, the digest A_i signed. No dropped, withheld, fabricated, duplicated, or altered chunk can therefore be certified complete.*

Proof. $\mathcal{B}$ simulates A_i with a signing oracle for Σ^A and runs $\mathcal{A}$, which controls the service and the store. Suppose the verifier returns COMPLETE or REPAIRED for sensor i.

Scope. Neither verdict is reached while some adapter lacks a final manifest in scope or a tail is pending, so a final manifest of A_i lies inside the checkpointed scope.

Declarations. The verifier checks the chain from $\mu_{i,0} = H(\texttt{0x09} \| \mathsf{enc}(e, i))$, link by link, against pk_i^A, which it holds from provisioning; the genesis check ensures that the history is anchored at the same inputs. An accepted manifest whose hash $\mu_{i,v}$ was never queried is a Σ^A forgery, valid while no index repeats. One whose hash was queried but whose body differs gives a collision in the `0x0a` call. By induction on v the accepted chain is therefore A_i's own chain, a skipped manifest having returned WITHHELD; it ends in A_i's final manifest, and nothing after it is accepted. A repeated declaration returns INVALID, and the declared numbers must be exactly $\{0, \ldots, f_i\}$. So $\mathcal{E}$ and every digest $H(c_{i,s})$ are fixed by the adapter, each $s \in \mathcal{E}$ once.

Committed leaves lie in $\mathcal{E}$, each once. The leaf identities of each window are reconstructed from its records, which D_t binds, and must number N_t. Every *miss* entry lies inside its own interval, is not in the auth set, and is not already missing; an auth entry below the interval only reclassifies an earlier missing gap; and a late arrival only resolves an existing gap. These checks stop a record from inventing a gap in order to re-commit a number later. Independently, an identity committed twice anywhere in the scope returns INVALID (line 33). A committed identity that is not declared returns INVALID when $s \le f_i$ or A_i has a final manifest (line 35), which holds here. Hence the committed identities of sensor i lie in $\mathcal{E}$, each once.

Every element of $\mathcal{E}$ is committed. The intervals tile $[0, \nu_i - 1]$. A declared number above $\nu_i - 1$ is added to the absence set as INGESTABSENT, and a declared number inside the intervals that is not a leaf sits in an unresolved gap and is also an absence. Either caps the verdict at ACCOUNTED.

Payloads. Neither verdict is reached without a store or with *pend* set, so every committed position has been opened with a path of length $\lceil \log_2 N_t \rceil$ to Λ_t. By Lemma S1 the opened preimage is the committed leaf, unless H collides. The verifier compares $H(c)$ with the declared $H(c_{i,s})$ and returns INVALID on a mismatch; a payload $c \neq c_{i,s}$ with the same digest is a collision, whoever produces it, so collusion between store and service does not help.

$\mathcal{B}$ outputs the forgery or the collision found in whichever step fails, which gives the bound. Collision resistance is used only for the `0x0a` manifest-body hash, the payload digest, and leaf and path binding; the theorem does not rely on Theorem 2, whose signer may now be malicious. Hiding the newest block leaves the scope without a FINAL manifest, or leaves committed leaves beyond the last manifest, which yields PREFIX or STALE rather than a complete epoch. Under a compromised service the difference between REPAIRED and COMPLETE rests on the service's late-arrival lists and is the service's claim; only the completeness of the declared record is guaranteed. □

This is the guarantee the extension adds over the conference version and over any design that authenticates only at the service: completeness then depends only on the adapters and on the keys the verifier holds from provisioning, and a fully compromised service cannot certify an incomplete subset of the declared record as complete.

Locating absences, and what it costs when the service lies. Under an honest service the four absence classes of Section 3 of the main article correspond to the four authenticated boundaries, and each is disjoint from the others by construction. A number in an unresolved $\mathcal{G}^{\text{miss}}$ was declared by the adapter and never arrived intact at ingestion; by Theorem 3 no committed history can hide it. A number in $\mathcal{G}^{\text{auth}}$ arrived only with invalid tags; under an honest adapter with a synchronized key this indicates modification or injection on the link, but the verdict states only the observable fact, an ingestion authentication failure, because adapter faults and key desynchronization produce the same evidence. A committed leaf without a receipt bit by $T_t + \theta_{\text{st}}$ was committed but never acknowledged by the store; the evidence does not separate a service that failed to deliver from a store that declined to acknowledge, and the verdict does not claim to. A receipted leaf whose chunk the store cannot return, or returns with a different leaf value, is a custody loss: Lemma S1 rules out a substitute that matches the committed leaf, and producing the receipt without the store requires a forgery against Σ_{st} under pk_{st}, which the verifier holds from provisioning and the genesis fixes. Under an honest service, attribution therefore needs no trust in the links, in the store, or in the party presenting the evidence, and it never asserts more than the last authenticated boundary a chunk crossed.

When the service itself may be compromised, this attribution weakens in specific places. The classes $\mathcal{G}^{\text{miss}}$ and $\mathcal{G}^{\text{auth}}$, and the arrival times behind them, are the service's own observations; a malicious service can relabel a dropped chunk as an authentication failure or move it between windows. HandoffAbsent becomes a service claim as well, because the service decides what it forwards to the store. CustodyLoss still rests on the store's signature, but only while the store does not collude with the service; a colluding store can sign or refuse receipts at the service's direction. What does not weaken is Theorem 4: whatever the service claims about *where* a chunk was lost, it cannot remove that chunk's identifier from the adapter-signed scope. The honest-service attribution is thus extra detail that only a trusted service provides, on top of a completeness guarantee that holds without it. Service-signed observations are claims, not verified facts, once the service is no longer trusted, and the conference attribution theorem does not carry over unchanged.

5 Attack coverage

Table S3 lists the 44 scenarios of the attack injector in its order, with the verdict that all eight trials returned with adapter-signed manifests. The last column gives the verdict of the MAC-only ablation, in which declarations are authenticated only by the MAC, where it differs; in every other scenario the two modes agree. †The manifest scenarios do not apply in MAC-only mode, which has no manifests; there the same mission is benign and returns Complete, and the scenario that omits manifests omits the service's gap entries instead. Where an absence class is shown, the injector also checked the exact absence set, except for the misbound receipt, where it checked the verdict and observed the class.

Table S3: Attack coverage: position, attack, detecting mechanism, and verdict, with the MAC-only ablation verdict where it differs.

Position	Attack	Mechanism	Verdict	MAC-only
None	benign mission	none needed	Complete	
None	retransmit five chunks	(i, s) uniqueness	Complete	
Link	drop one chunk	heartbeat, interval tiling	Accounted (IngestAbsent)	
Link	drop a burst of chunks	heartbeat, interval tiling	Accounted (IngestAbsent)	
Link	silence chunks and heartbeats of one sensor	staleness flag	Stale	
Link	modify a payload	chunk tag	Accounted (IngestAuthFail)	
Link	inject forged chunks and heartbeats	chunk and heartbeat tags	Complete	
Link	replay old chunks	(i, s) uniqueness	Complete	
Link	reorder within a window	leaves in (i, s) order	Complete	
Link	delay a chunk across a window	late arrival resolves the gap	Repaired	
Publication	delete a window	window index, chaining	Invalid	

Position	Attack	Mechanism	Verdict	MAC-only
Publication	insert a window	window index, chaining	Invalid	
Publication	swap two windows	window index equals position	Invalid	
Publication	truncate the tail	checkpoint deadline	Withheld	
Publication	substitute a record, recompute the chain	checkpoint signature, Theorem 2	Invalid	
Publication	forge a checkpoint	signature under pk	Invalid	
Custody	withhold a receipt	receipt deadline θ_{st}	Accounted (HandoffAbsent)	
Custody	corrupt a chunk on its way to the store	store checks against Λ_t; no receipt bit	Accounted (HandoffAbsent)	
Custody	delete a stored chunk	receipt, opening	Accounted (CustodyLoss)	
Custody	substitute a stored chunk	leaf and path binding, Lemma S1	Accounted (CustodyLoss)	
Link	drop a burst, then resend	buffer, late arrival	Repaired	
Link	modify a payload, then resend	late arrival resolves the auth gap	Repaired	
Link	drop a burst; first resends lost too	repeated request	Repaired	
Link	drop a burst; suppress resend requests	gap stays unresolved	Accounted (IngestAbsent)	
Link	forge resend requests	resend is a duplicate	Complete	
Service	erase tail gaps, re-sign	interval tiling	Accounted	
Service	fabricate chunks, re-sign	manifest digest, Theorem 4	Invalid	Complete
Service	omit manifests and gaps	manifest chain	Withheld	Invalid
Service	hide the newest blocks	leaves beyond the last manifest; no Final	Stale	Complete
Link	duplicate many chunks	(i, s) uniqueness	Complete	
Custody	receipt bound to another window	(e, t, N_t, Λ_t) in the receipt	Accounted (HandoffAbsent)	
Service	fake gap, then fabricated late leaf	gap outside its interval	Invalid	
Service	fake gap, then duplicate leaf	gap outside its interval	Invalid	
Service	forge the unreceipted last window	every position opened	Invalid	Repaired
Service	substitute the genesis	genesis check	Invalid	
None	honest history, late audit	schedule ends at the shutdown checkpoint	Complete	
None	honest history, late audit, shutdown on a scheduled checkpoint window	shutdown checkpoint signs $end = 1$	Complete	
Publication	audit before the last checkpoint is due, checkpoint not yet published	checkpoint pending until its deadline	Prefix	
Link	drop a manifest, resend path intact	manifest re-requested; later ones held in order	Complete	Complete†
Link	drop a manifest and every request	frontier stops; later chunks unconfirmed	Stale	Complete†

Position	Attack	Mechanism	Verdict	MAC-only
None	honest history, shutdown on a scheduled checkpoint window, audited on time	shutdown checkpoint signs $end = 1$	COMPLETE	
Publication	withhold the shutdown checkpoint when it falls on a scheduled window	checkpoint overdue	WITHHELD	
Service	sign an end checkpoint in mid-history and keep publishing	end flag valid only on the last window	INVALID	
Custody	sign a receipt whose bitmap is too short	malformed receipt acknowledges nothing	ACCOUNTED (HANDOFFABSENT)	

6 Communication and key lifetime

Communication. The verifier receives, per window, the header $h_t = (t, N_t, \Lambda_t, D_t)$ (80 bytes raw, 96 bytes as a standalone enc), the m encoded records, and $1/k$ of a checkpoint. A record is not fixed in size. In the reference encoding each gap or late-arrival number adds 9 bytes, each receipt-acknowledged pair (t', s) in $\mathcal{Q}_{i,t}$ adds 23 bytes, and each manifest carried in a record adds one signature, 72 bytes of fixed fields, and 51 bytes per declared chunk, so record traffic grows linearly with the chunk rate unless consecutive numbers are range-encoded. An XMSS signature with $n = 32$-byte hashes, Winternitz parameter $w = 16$, and tree height h has $4 + n + (len + h)\,n$ bytes with $len = 67$ [4, 5], that is 2500, 2692, and 2820 bytes for $h = 10, 16, 20$; the published pair (t, R_t) adds about 40 bytes per checkpoint. The upstream link carries one tag per chunk and one heartbeat per Δ_h, a 32-byte tag being negligible against 8 KiB chunks. Receipts travel between store and service and add one signature and one N_t-bit map per window to the evidence the verifier receives; the store's signing cost stays off the critical path of Eq. (4) of the main article, because each receipt enters the record of whichever window it arrives in.

Table S4: XMSS key lifetime and checkpoint bandwidth at $\Delta = 10\,\mathrm{s}$. The bandwidth column counts checkpoint signatures only.

h	Signature (B)	k	Lifetime Ω	Bandwidth (B/s)
10	2500	1	2.8 h	250.0
10	2500	6	17.1 h	41.7
16	2692	1	7.6 d	269.2
16	2692	6	45.5 d	44.9
20	2820	1	121.4 d	282.0
20	2820	6	728.2 d	47.0

6.1 Evidence-cost model

Fig. 3(c) of the main article counts, per telemetry window of the live run, what a verifier must receive besides the telemetry, averaged over the 32 telemetry windows: 4070 chunks and 5,555,936 payload bytes in all, that is about 127 chunks and 173,623 bytes per window. Every mechanism is counted from its binary fields, and every signature is XMSS-SHAKE256 with $h = 10$, 2500 bytes.

- *Merkle log* (RFC 6962 [6]): one signed tree head per window, $1 + 1 + 8 + 8 + 32 = 50$ bytes of signed fields (version, signature type, timestamp, tree size, root), a 2-byte signature length, and the signature, 2552 bytes.
- *SDLS with CFDP* [7, 8]: an authentication-only security association adds a 2-byte SPI, a 4-byte sequence number, and a 16-byte MAC to every TM frame with a 1105-byte data field, about 157 frames per window, plus about 310 bytes of CFDP class-2 PDUs per window, 3767 bytes.
- *Signed syslog* (RFC 5848 [9]): one signature block per source and window, about 40 bytes of fields plus a signature, and a 32-byte hash per message, 9150 bytes. This is the binary equivalent; the base64 text encoding of RFC 5848 adds a third.

- *NEITH-ACE*: the manifests (one signature, about 40 bytes of fields, and 32 bytes per chunk; 66 manifests over 32 windows), one store receipt (a signature, the $\lceil N_t/8 \rceil$-byte bitmap, and 20 bytes), the records (about 300 bytes), the 80-byte header, and a $1/k$ share of a checkpoint (signature plus about 40 bytes, $k = 6$), 12,648 bytes.

The totals are 2.5, 3.7, 8.9, and 12.4 KiB per window, or 1.5, 2.2, 5.3, and 7.3% of the telemetry. The reference implementation's own encoding measures 17.7 KiB per window, 10.4% of the telemetry: its encoded records (482,510 bytes over all 34 windows of the run, divided by the 32 telemetry windows like every other bar) length-prefix every field and repeat the receipts' acknowledgements, and the receipt, the 96-byte encoded header, and the checkpoint share come on top. Nitro [10] has no bar because its symmetric tags are not evidence a third party can check. Signed syslog with freshness and DASLog [11] have no bar of their own: the comparison of Table 4 in the main article reimplements both with the adapters' manifests (docstring of `sota.py`), so their source evidence is the manifest part of the NEITH-ACE bar.

7 Parameter margins against a quantum adversary

Generic quantum collision search on a 256-bit hash output needs about 2^{85} queries, fewer than the 2^{128} of classical birthday search, but the known algorithms that reach this count also need quantum memory or hardware of comparable size; whether they would beat classical search in practice depends on those resources, and the 256-bit output keeps either cost far out of reach. Grover search reduces exhaustive search for a 128-bit MAC key to about 2^{64} sequential steps, which NIST still places in its lowest post-quantum security category; a deployment that wants more margin can use KMAC256 with a 256-bit key in place of KMAC128. XMSS-SHAKE256 with 256-bit parameters relies only on properties of SHAKE256, which these generic attacks do not break at that output length [4, 5, 12].

8 Reproducing the scheme and the experiments

This section records every choice an independent implementation needs in order to produce the same bytes and the same experiments as the reference implementation; the code itself is at `https://github.com/mealycpp/neith-ace`.

Encoding. The encoding $\mathsf{enc}(x_1, \ldots, x_n)$ is the field count n as a 4-byte big-endian integer, followed by each field with a one-byte type marker. An integer is `i` and 8 bytes, signed, big-endian; a Boolean is `b` and one byte, `0x01` or `0x00`; a byte string is `y`, its length in 4 bytes, and its bytes; a text string such as `"chunk"`, `"hb"`, `"mf"`, or `"rcpt"` is `s`, its UTF-8 length in 4 bytes, and its bytes; a list or tuple is `l` followed by its own `enc`; an empty value is `n`. The type marker and the counts make the encoding injective and prefix-free. A chunk's leaf is $H(\texttt{0x00} \| \mathsf{enc}(e, t, i, s) \| c)$, with the payload c appended raw. The configuration `cfg` in the genesis root is the list of the epoch, Δ, γ, Δ_h, κ, k, θ, θ_{st}, the names of the hash, the MAC, and the signature parameter set, the declaration mode, and Δ_m; the real-valued times are written as decimal strings such as `"10.0"` and `"0.5"` (the shortest decimal that reads back as the same binary value), κ and k are integers, and the names are `"shake128"`, `"kmac128"` or `"ascon-cxof128-keyed"`, `"XMSS-SHAKE256_10_256"`, and `"signed"` or `"mac"`. The genesis root is $H(\texttt{0x06} \| \mathsf{enc}(e, \mathsf{enc}(\mathsf{cfg}), pk, \mathsf{apk}, \mathsf{spk}))$, where each public key is its 68-byte XMSS encoding (the 4-byte parameter identifier, the root, and the public seed), $\mathsf{apk} = \mathsf{enc}$ of the tuples $(i, \text{parameter name}, pk_i^A)$ in sensor order, and $\mathsf{spk} = \mathsf{enc}(\text{parameter name}, pk_{\mathrm{st}})$. The main article numbers sensors $i \in \{1, \ldots, m\}$; on the wire, and in every encoding, sensor i is carried as the integer $i - 1$, so the encoded identifiers are $0, \ldots, m - 1$. This fixes the bytes of $\mu_{i,0}$, the genesis, the records, the leaves, and the testbed keys. A record is the tuple $(i, a, b, \mathcal{G}^{\mathrm{miss}}, \mathcal{G}^{\mathrm{auth}}, \mathcal{L}, \mathcal{Q}, r, \hat{s}, u, \phi, \mathcal{M})$ with every set written as an ascending list, $\mathcal{Q}$ as a list of pairs (t', s), ϕ as a Boolean, and $\mathcal{M}$ as a list of manifests $(v, \hat{s}, \mu_{v-1}, \mathit{fin}, \Pi, \text{signature})$ in index order, Π a list of pairs $(s, H(c))$; a window's records are encoded as $\mathsf{enc}(\rho_{1,t}, \ldots, \rho_{m,t})$, and $r_{i,t} = H(\texttt{0x08} \| \mathsf{enc}(x_1, \ldots))$ over the byte strings $y = \mathsf{enc}(t', q_{t'}, \beta_{t'})$ of the receipts that arrived, in arrival order, so $r = H(\texttt{0x08} \| \mathsf{enc}())$ when none did. A heartbeat tag covers $\mathsf{enc}(\texttt{"hb"}, e, i, \hat{s}, u)$.

Primitives. H is SHAKE128 with a 32-byte output; where Ascon-XOF128 is used instead, H is the first 32 bytes of its output. The MAC is either KMAC128 with a 16-byte key, a 32-byte output, and the customization string `"NEITH-ACE"`, or keyed Ascon-CXOF128, computed as Ascon-CXOF128 over the 16-byte key followed by the message, with customization string `"NEITH-ACE-MAC"` and a 32-byte output. The host pipeline, and so every end-to-end and injector result, uses KMAC128; only the F′ adapter uses keyed Ascon-CXOF128. SP 800-232 standardizes Ascon-CXOF128 as an extendable-output function, not as a MAC, so the theorems assume only that the chosen MAC is unforgeable and do not prove it for this construction. The closest proven result is the outer-keyed sponge of Andreeva, Daemen, Mennink, and Van Assche [13]: a sponge that starts from the all-zero state and absorbs a uniformly random key, a whole number of rate blocks long, before the message. With an ideal b-bit permutation, capacity c, rate r, key length k, M construction queries, N offline permutation queries, and multiplicity $\mu \le 2M$, its

single-key distinguishing advantage is at most $(M^2 + 2\mu N)/2^c + \lambda(N) + 2(k/r)N/2^b$, with $\lambda(N) \le N^2/2^{c+1} + N/2^k$ when $k \neq r$. Keyed Ascon-CXOF128 meets some of these hypotheses and not others. Its 128-bit key is two whole 64-bit rate blocks ($k/r = 2$), and the padding is injective. But it does not start from the all-zero state: SP 800-232 initializes it as $\mathsf{p}^{12}(IV \| 0^{256})$ and then absorbs a block holding the customization length and the customization string, padded to whole blocks, so the key is absorbed after a fixed public prefix rather than first. For an ideal permutation that prefix only replaces the zero starting state with another public one, and we expect the bound to carry over, but the cited theorem does not state this, and Ascon's permutation is not ideal. We therefore claim no bound for this construction. Its unforgeability is an assumption, motivated by that result; if the result did carry over, with $b = 320$, $c = 256$, $r = 64$, and $k = 128$ its dominant term would be $N/2^{128}$. A deployment that needs a standardized MAC uses KMAC128 in the adapter as well. Every signature is XMSS-SHAKE256_10_256, the parameter set of NIST SP 800-208 for the RFC 8391 construction, computed with the xmss-reference implementation and its BDS traversal; each signer persists its updated secret state to storage, and synchronizes it, before it releases a signature. A manifest signature covers the 32-byte value μ_v, a checkpoint signature covers $H(\texttt{0x07} \| \mathsf{enc}(e, t, \mathit{end}) \| R_t)$ with the Boolean *end* set only on the shutdown checkpoint, and a receipt signature covers $\mathsf{enc}(\texttt{"rcpt"}, e, t, N_t) \| \Lambda_t \| \beta_t$. The heartbeat counter u starts at 1, so the first heartbeat carries $u = 1$. Leaves are ordered by sensor and then sequence number, and bit j of the receipt bitmap, counted from the least significant bit of the first byte, stands for leaf position j.

Service and adapter rules. At each close the service sets b to the largest of the previous b, the $\hat{s}$ of the latest authenticated heartbeat received before T_t, and the highest number received in the window. A heartbeat with a bad tag or a counter not above the last accepted one is discarded, a bad tag on a number already received is ignored, and a bad tag on a number above b waits, for at most 1024 numbers, until a declaration covers it. After every close except the last, the service requests every unresolved gap, up to 256 per sensor, and every manifest index below the highest one seen that has not arrived; there is no separate retry rule, because an unanswered request is simply repeated at the next close. Each adapter keeps its last $B = 4096$ chunks and every manifest of the epoch for resending.

Capacity model. Eq. (5) of the main article gives an analytical sizing estimate, not a measurement; Fig. 3(b) of the main article reports the measured whole-pipeline capacity instead (Table S5). For reference, evaluating Eq. (5) with the Pi 4 medians $c_{\mathsf{MAC}} = 114.3\,\mu$s and $c_H = 47.7\,\mu$s for 8 KiB, $c_{\mathrm{n}} = 4.2\,\mu$s, $m\,c_\rho = 0.49$ ms for $m = 2$, $c_{\mathrm{root}} = 9.2\,\mu$s, and the worst signing time with its state write, $c_{\mathrm{sig}} = 41.1$ ms, at $\Delta = 10$ s and $\gamma = 0.5$ s, with $N_{\max} = \lceil \lambda\Delta \rceil$ solved as a fixed point, gives 5690 chunks/s for one worker ($P = 1$) and 12,962 for $P = 2.28$, the measured ratio of SHAKE128 throughput on 8 KiB inputs with four processes and with one (367.1 against 161.2 MB/s). These estimates cover the service's work alone. The measured 1597.7 chunks/s is lower because the reference service is single-threaded and, in the sweep, shares the Pi's four cores with two adapters and the store; it is the figure the main article reports.

Live testbed. The capture adapters on the Pi 4 read the u-blox receiver as one chunk per NMEA sentence and the LiDAR serial stream as one chunk per 8192-byte block, and send them over TCP to the integrity service on a Pi Zero 2 W. In the testbed each MAC key K_i is the first 16 bytes of SHAKE256 applied to a shared setup secret, the string `"adapter"`, and i as a 4-byte big-endian integer; the adapters' XMSS keys are generated locally by each adapter process. Link loss is injected at the service's input: the first transmission of each chunk is discarded independently with probability 0.05, and resends are never discarded. The cheating store alters, in every window with $t \equiv 1 \pmod 3$ (windows $1, 4, \ldots, 31$, eleven in all), after signing that window's receipt, the payload of the oldest stored GNSS chunk it has not yet altered, replacing it with 64 random bytes; the oldest stored chunk is always a GNSS chunk, because sensor 0 sorts first. A run starts with a warm-up in which the adapters connect, before window 0, then has 32 telemetry windows, one drain window that collects stragglers, and one settlement window that waits up to θ_{st} for outstanding receipts and ends with a checkpoint, after which the verifier audits the epoch. The parameters are those of Section 6 of the main article. The GNSS chunks were single NMEA sentences of about 58 bytes at about 10.7 per second, and the LiDAR chunks 8192-byte blocks at about 2 per second. The live run was recorded before two later changes to the reference implementation, the *end* flag in the shutdown checkpoint and the in-order commitment of manifests; neither affects it, because its shutdown checkpoint fell on window 33, not on a scheduled checkpoint window, and no manifest was lost.

Attack injector and comparison. Each simulated mission has two sensors, 12 telemetry windows (11 in the four scenarios that put shutdown on a scheduled checkpoint window), and one settlement window that ends with the shutdown checkpoint, with 8192-byte chunks. A sensor with rate λ_i (5.529 and 1.432 chunks per second, the rates measured in the conference paper) emits its first chunk at a time drawn uniformly from $[0, 1/\lambda_i)$, and each later chunk after a gap of $0.5/\lambda_i$ plus an exponential variable of mean $0.5/\lambda_i$, so its mean rate stays λ_i while its arrivals

are bursty; chunks stop 0.05 s before the end of the last telemetry window. Heartbeats start at a phase drawn from $[0, \Delta_h)$, manifests are sent at $\Delta_m - 0.2 + j\Delta_m$ s and the final one 0.02 s before the end, every message crosses the link with a delay drawn from $[4, 6]$ ms, the MAC keys are drawn from the mission's generator, and the audit runs at $\Delta(W+1)+\gamma+\theta$ unless a scenario sets another time. Heartbeats, manifests, window closes, and deadlines follow the same parameters as the live run in simulated time, and all cryptography is real. Each mission is driven by Python's `random.Random` seeded with the string formed from the base seed 2026, the scenario name, and the trial number joined by hyphens, so a scenario and trial reproduce exactly; the attack's own parameters, such as a burst of 2 to 8 consecutive chunks placed between the second window and the third-to-last, are drawn from the same generator, and the exact rule for each scenario is in the injector's source. Processing takes no simulated time. The injector deliberately uses the conference paper's rates and 8 KiB chunks for both sensors, a heavier evidence load than the live run's short GNSS sentences. The attack table above uses eight trials per scenario in each declaration mode; the comparison of Table 4 in the main article runs the same generator for twenty missions per cell, and each baseline verifier sees only the evidence its mechanism would hold. A baseline that detects an incomplete record but cannot say at which step a chunk went missing scores D; one that attributes each absence scores A. Signed syslog with freshness flags a source as stale when its last signature block is older than the mission end minus $1.5\Delta_m$. When the newest windows are hidden, NEITH-ACE returns STALE if the last delivered window committed a chunk after that window's manifest, and PREFIX otherwise, which happened in 2 of the 20 comparison missions and none of the 8 injector trials.

What produces each result. The Pi 4 microbenchmarks time 300 repetitions of each hash and MAC on 8 KiB inputs and 200 XMSS signatures including the state write. Fig. 3(a) and the repair and custody results come from the per-window log of the live run. Fig. 3(b) comes from `capacity.py`, which runs `live.py` on one node with two constant-rate adapters (`paced` source, 8192-byte chunks), $\Delta = 10$ s, $\gamma = 0.5$ s, $k = 1$, four telemetry windows, and signed manifests; it doubles the offered rate from 100 chunks/s until a rate fails one of the three conditions in the main article or would exceed the free memory, then bisects, and records each run's per-window log and verdict. A run passes when the verifier returns COMPLETE, every window meets the no-stutter rule, and the adapters' delivered rate, the chunks they produced divided by their running time, is at least 95% of the offered rate. The criterion concerns the producers only: in every run, including the failing ones, every chunk the adapters produced was committed, stored, and verified, so no run lost data. A shortfall means the adapters, sharing four cores with the service and the store, could not produce chunks as fast as offered. Table S5 lists the sweep; the highest delivered rate that passes, 1597.7 chunks/s (13.09 MB/s), is the capacity reported in Section 6 of the main article. The 3200 chunks/s step was skipped because its chunks would not fit in the Pi's free memory. The attack table comes from the injector, Table 4 of the

Table S5: Capacity sweep on the Pi 4, in the order run (8192-byte chunks, two adapters, $k = 1$, four telemetry windows per run).

Offered (chunks/s)	Delivered (chunks/s)	Delivered (%)	Worst $E_t - T_t$ (s)	Verdict	Passed
100	99.9	99.9	0.65	COMPLETE	yes
200	199.9	100.0	0.76	COMPLETE	yes
400	399.9	100.0	0.96	COMPLETE	yes
800	799.8	100.0	1.32	COMPLETE	yes
1600	1522.7	95.2	2.50	COMPLETE	yes
2400	1512.5	63.0	4.39	COMPLETE	no
2000	1559.5	78.0	3.85	COMPLETE	no
1800	1603.0	89.1	2.37	COMPLETE	no
1700	1567.4	92.2	2.17	COMPLETE	no
1650	1597.7	96.8	2.22	COMPLETE	yes

main article from the comparison runs, and Fig. 3(c) from the evidence-cost model above. The verifier was checked only through these scenarios, the edge-case tests in the repository's `tests` directory (a receipt not yet due, malformed receipts, and altered or missing openings), and the proofs; it has not been formally verified. The F′ adapter's byte compatibility comes from a test program that computes a tag, a digest, a manifest genesis, and a manifest signing message on fixed inputs, signs the last with XMSS and verifies the signature, and compares the four values with the Python reference; it gave identical output on the Pi 4 and on an x86 host.

References

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

[2] 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.

[3] 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.

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

[5] 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.

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.