Signed Is Not Safe: Measuring the Boundary Between Cryptographic Integrity and Semantic Trust in AI Supply Chains

Karan Deshmukh, Sudheer Reddy Jogi, Dola Krishna Dama, Rambabu Mekala
MSc Cybersecurity and Data Science, ESAIP École d'Ingénieurs, Angers, France
Projet Applicatif 2 (Applied AI Projects), supervised by Sofiane Hamrioui
Corresponding author: domail2karandeshmukh@gmail.com

***Abstract*—A deployed model is the product of datasets, pre-trained weights and fine-tuning steps from different parties, yet software supply-chain controls sign one build or one file and say nothing about this lineage or about what a valid signature means for a model's behaviour. We present AI-ChainGuard, which gives every artifact of an AI pipeline, from dataset to deployment record, a signed manifest whose payload embeds the manifest digests of its parents. The result is a hash-linked provenance graph that a verifier validates with ten checks. Trust and revocation live in a signed, pinned, append-only log, and a deployment gateway verifies the whole chain at deployment time before loading a model from the verified bytes. We then *measure* the boundary of such guarantees with a reproducible suite of 48 attack and control variants in seven experiments. Every post-signing manipulation in the suite, from a 10^{-6} change to one weight to the truncation of the trust log, is detected in 1.7–15.1 ms including the chain. But a dataset poisoned *before* signing verifies and deploys with a 77 % backdoor success rate behind a VERIFIED status, an adaptive trigger evades content heuristics at 51 %, and a trusted publisher's backdoored model verifies with 100 %. Attestation costs 0.8 % of training time, 8.5 ms per full-chain verification, nothing per inference and 1.4–3.1 KB per manifest, and exports as in-toto/DSSE attestations with ChainGuard or SLSA predicates. We argue that AI supply-chain controls must report cryptographic and semantic evidence separately, and release code, tests and scenarios.**

***Index Terms*—AI supply chain, attestation, backdoor, data poisoning, digital signatures, in-toto, model provenance, revocation, SLSA, software supply chain security.**

I. Introduction

Machine-learning systems are assembled rather than written: a production model descends from a dataset collected by one team, a pretrained model downloaded from a public hub, a fine-tuning run on someone else's infrastructure and a packaging step, and every hand-over is an opportunity for substitution or manipulation. In December 2022 a dependency-confusion attack replaced `torchtriton`, a dependency of the PyTorch nightly builds, by a package that exfiltrated SSH keys and files [1]. In February 2024 JFrog found about one hundred Hugging Face models whose pickle serialisation executed code on load, one opening a reverse shell [2]; in 2025 ReversingLabs described models whose deliberately broken pickle stream ran its payload before the scanner failed [3]. Poisoning is cheap: a 2025 study backdoored language models of 600 M to 13 B parameters with a near-constant 250 poisoned documents [4], and Carlini et al. showed that web-scale datasets can be poisoned for tens of dollars [5]. Article 15(5) of the EU AI Act now requires measures against "attacks trying to manipulate the training data set (data poisoning), or pre-trained components used in training (model poisoning)" [6]; the NIST AI RMF [7] and MITRE ATLAS technique AML.T0010, *AI supply chain compromise* [8], frame the same risk.

The natural reaction is to reuse software supply-chain security: SLSA [9], in-toto [10] and Sigstore [11] give builds a verifiable provenance, and OpenSSF Model Signing extends Sigstore to model directories [12]. Two gaps remain. These tools attest a build or a file: nothing links the dataset to the base model, the base model to the fine-tuned model and that model to what was deployed, with a policy on who may sign which step and a revocation that propagates along the links. And a signature proves that bytes are what a key holder attested, yet users of model hubs are tempted to read *signed* as *safe*; for source code that reading is tolerable because code can be reviewed; for a tensor of weights it is not, and, to our knowledge, nobody has measured how far it is wrong.

This paper contributes (1) AI-ChainGuard, an attestation chain across four pipeline stages with a recursive verifier of ten checks, a role-bound signer policy, a deployment gateway and a signed, pinned, append-only *trust log* that removes the database from the trusted base for keys and revocations; (2) a reproducible suite of 48 attack and control variants in seven experiments, from post-signing tampering and provenance rewriting to revocation bypasses against the log itself, untrusted publishers and poisoning before signing; (3) a measurement of the boundary: every post-signing manipulation is detected, while signed poison verifies with a 77–100 % backdoor success rate and content heuristics are evaded; (4) the cost of attestation and its interoperability through in-toto/DSSE and SLSA exports.

II. Background and Related Work

A. Threats along an AI pipeline

We distinguish threats by *when* they act. After an artifact is signed, an attacker with access to storage, a mirror or

the network can modify bytes, substitute an artifact, rewrite the recorded ancestry so that a model appears to descend from an approved dataset, forge or replay signatures, use a compromised or revoked key, or tamper with the system that records who is trusted and what is revoked; all of this is, in principle, within reach of cryptography. Before signing, an attacker who controls the data source, the publisher of a pretrained model or the training host plants a backdoor [13] that a legitimate signer attests in good faith. Malicious serialisation [2], [3] sits in between: the file is exactly what was signed and executes code when loaded.

B. Signing, provenance and transparency

Sigstore replaces long-lived keys by short-lived certificates for OIDC identities (Fulcio) and records every signature in a public transparency log (Rekor) [11]. in-toto describes a supply chain as a layout of steps with authorised functionaries and verifies each step's link metadata [10]; its attestation framework defines the *Statement* (subject digests plus a typed predicate) and the DSSE envelope used by Sigstore and GitHub artifact attestations [14], [15]. SLSA v1.1 (April 2025) defines build-provenance levels and the provenance predicate consumed by policy engines [9]; OpenSSF Model Signing v1.0 (April 2025) hashes every file of a model directory into one signed manifest with Sigstore or PKI signatures [12]. The Update Framework pins a root role and protects against rollback with versioned, signed metadata [16]; Certificate Transparency makes a log tamper-evident with a Merkle tree [17]. Model cards [18] and datasheets [19] describe artifacts without signing them; safetensors removes code execution from loading [20]. Content defences such as Neural Cleanse [21], STRIP [22], spectral signatures [23] and activation clustering [24] detect some backdoors and are evadable by adaptive attackers. Table I positions AI-ChainGuard: it does not replace these tools; it links the artifacts they sign, enforces who may sign what, and reports what a signature does and does not establish.

III. System Design

A. Threat model

Trusted: the organisation's signing keys (including the root key), the small pin file that anchors the trust log, the supply-chain policy and the verifier code. Treated as hostile: artifact storage, the index database, datasets, external model repositories, training hosts and the network. An attacker may read and write any artifact, manifest, index row or log record, replay or forge envelopes, hold a revoked key or publish models under an unknown identity. Out of scope are compromise of the trusted components, side channels, and the correctness of the training code. The verifier therefore recomputes every hash, digest and Merkle root instead of believing stored values, and reads nothing from the database that it could not confirm otherwise.

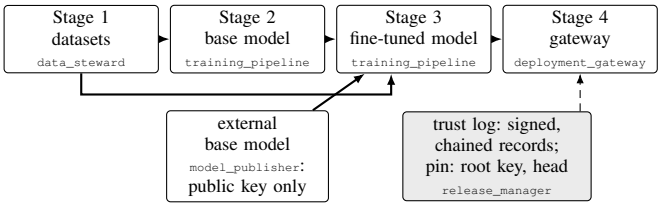


Fig. 1. Four attested stages and the signing role of each. Each child's signed payload embeds the manifest digests of its parents (solid arrows). Internal base models require a dataset parent; an external base model has no internal lineage and may be signed only by a registered publisher key. The gateway re-verifies the whole chain; its verification, like every other, replays the signed trust log (dashed).

B. Manifests and the hash-linked chain

Every artifact receives a manifest body m: its type and pipeline stage, the SHA-256 and size of the file, for CSV datasets the Merkle root over the hashes of its rows, the list of parents, free-form provenance (training configuration, environment), the results of pre-attestation content checks, and the signer's identity, role and key ID. The body is serialised in a canonical JSON form (sorted keys, no insignificant whitespace) and signed as a whole with Ed25519 [25]:

$$\sigma = \mathrm{Sign}_k\big(\mathrm{canon}(m)\big), \qquad d(m) = \mathrm{SHA256}\big(\mathrm{canon}(m)\big). \tag{1}$$

Signing the whole body rather than the file hash alone prevents an attacker from keeping a valid (hash, signature) pair and changing the stage, the parents or the signer role around it. The manifest digest $d(m)$ is the artifact's identity in the provenance graph: a child lists each parent as $(\mathrm{id}_p, d(m_p), \mathrm{relation})$ inside its own signed body, so that

$$d(m_{\mathrm{child}}) \text{ commits to } d(m_p) \text{ for every ancestor } p, \tag{2}$$

as a Git commit commits to its parents. Rewriting an ancestor, even re-signing it with the legitimate key, changes $d(m_p)$ and breaks every descendant. For datasets, the Merkle tree (RFC 6962 construction [17] over line hashes) lets the verifier report *which* rows changed, authenticated by the signed root; the leaf hashes are kept in a sidecar that is trusted only if it reproduces that root.

C. Verification: ten checks and three outcomes

Verification of an artifact runs ten checks and recurses into its parents (Table II). The outcome is derived, never set: FAILED if any check fails, WARNING if integrity holds but a content check flagged the data here or upstream, VERIFIED otherwise. VERIFIED means *exactly what trusted, unrevoked signers in authorised roles attested, through an intact ancestry*; the system never upgrades it to "safe". The policy file states which role may sign which stage (Fig. 1) and which parent types each stage requires or forbids: an internal base model must descend from a dataset, a fine-tuned model from a dataset and exactly one base model of either kind, and the training key may not sign the external-import stage, so an internal model cannot be relabelled "external" to escape the dataset rule. Every report also records the sequence number and head

TABLE I
HOW AI-CHAINGUARD DIFFERS FROM EXISTING APPROACHES (CAPABILITIES AS DOCUMENTED BY EACH PROJECT, NOVEMBER 2026; "NO" MEANS NOT A GOAL OF THAT TOOL). NO COLUMN REACHES "YES" FOR POISONED CONTENT: THAT IS THE BOUNDARY THIS PAPER MEASURES.

	Sigstore / cosign	OpenSSF Model Signing	in-toto layouts, SLSA provenance	Model cards, ML-BOM, registries (MLflow, DVC)	Backdoor scanners	**AI-ChainGuard**
Scope	one artifact (OCI image, blob)	a model directory	steps of a software build	description, inventory, versions	model behaviour	**dataset → base → fine-tune → deployment**
Links stages	no	no	layout links steps; build inputs recorded	metadata lineage	no	**yes: parent manifest digests inside each signed payload**
Who may sign what	identity policies	signer identity	functionaries per step; builder identity	platform access control	n/a	**role per stage, verified (V4, V8); external publishers**
Revocation	certificate expiry, log	Sigstore ecosystem	key revocation in layout	stage transitions	n/a	**signed, pinned, append-only log; propagates to descendants**
Deployment decision	admission controllers	no	verification of the final product	no	no	**gateway: verify at deployment, safe load, smoke test, signed record**
Poisoned content	no	no	no	no	partial, heuristic	partial: heuristics give WARNING only; **measured, not claimed**

TABLE II
THE TEN CHECKS OF A VERIFICATION

	Check	Fails when
V0	Envelope	missing, malformed, or swapped (stored under another id)
V1	File hash	SHA-256 or size differs from the signed values
V2	Dataset Merkle root	rows changed (the report lists which lines)
V3	Ed25519 signature	invalid over the canonical body, or key not in the signed trust log
V4	Signer trust	key revoked, role or identity mismatch, role not allowed for the stage, index and log disagree
V5	Parent links	a parent envelope is missing or its recomputed digest differs from the embedded one
V6	Ancestry	any ancestor FAILED (recursive)
V7	Revocation	artifact or ancestor revoked in the signed log, index row deleted, or index flag cleared against the log
V8	Policy	wrong type for stage, missing or forbidden parent types, parent from a later stage
V9	Content flags	a signed pre-attestation content check flagged the data here or upstream: WARNING only

hash of the trust log it was verified against, and names the external artifacts whose provenance ends at a publisher.

D. The signed trust log

In the previous version (v0.3, October 2026) the trust store and the revocation flags lived in the SQLite index, and our own revocation experiment showed the consequence: an attacker able to write that file could clear a `revoked` flag and the withdrawn model verified again. Trust and revocation are now records of an append-only log. Record i carries a sequence number, the hash of record $i-1$, a timestamp, an action (`trust_key`, `revoke_key`, `revoke_artifact`, `rotate_root`) and its subject, and is signed by the root key k_R of the `release_manager` role:

$$r_i = \big(e_i,\ \mathrm{Sign}_{k_R}(\mathrm{canon}(e_i))\big), \qquad h_{i-1} \in e_i, \qquad (3)$$
$$h_i = \mathrm{SHA256}\big(\mathrm{canon}(r_i)\big).$$

A pin file in the trusted keys directory holds the root public key and the pair (n, h_n) of the last record. The verifier replays the log at every verification: each signature must verify under the active root (a `rotate_root` record signed by the old root introduces a new one), each record must chain to the previous, and the log must end exactly at the pinned head. A truncated log (rollback), an edited record, a deleted or substituted log, or a key present in the index but absent from the log, all make V3, V4 and V7 fail closed, and an index that disagrees with the log is reported as tampering. The structure is a single-writer hash-chained log (small enough to need no Merkle tree) with TUF-style root pinning and rollback protection [16]; publishing the records to a shared log such as Rekor so that all verifiers see one history is the natural next step. There is no un-revoke action: a withdrawn artifact is replaced by a new version.

E. Deployment gateway and external publishers

The gateway is the last line of defence. It verifies the full chain at deployment time (once per deployment, never per inference), refuses a FAILED model and, by default, a WARNING one, and refuses pickle-based formats. It reads the model bytes once, re-hashes them and loads the model from those same bytes, so that there is no time-of-check/time-of-use gap, runs a smoke test on held-out data, computes a release root (the Merkle root over the manifest digests of the model and its ancestors) and signs a deployment record that also binds the trust-log head it verified against. A model from an outside publisher enters through a separate path: the publisher signs a manifest with its own key, an administrator registers the publisher's *public* key out of band, and the import refuses anything unsigned or structurally wrong, stores the rest and verifies it; a failed import is kept as quarantined evidence that cannot be used as a parent or deployed. Its provenance ends, by design, at the publisher's identity.

F. Interoperability

Any attestation exports as an in-toto Statement v1 inside a DSSE envelope [14], [15], signed over the pre-authentication encoding (PAE) with the role key that signed the original manifest, with either a ChainGuard predicate (the full manifest) or a SLSA Provenance v1 predicate [9] in which the training or fine-tuning stage is the build, the dataset and base model are

resolved dependencies, the signing role is the builder and the manifest digest and Merkle root travel as by-products. The export is verified by an independent DSSE verifier in the test suite, including the specification's PAE test vector.

IV. Implementation

AI-ChainGuard is about 8 800 lines of Python 3.11+ (PyTorch, scikit-learn, FastAPI, Streamlit, the `cryptography` library) plus 1 600 lines of tests; one core library holds all security logic, and the CLI, REST API, ML pipeline, gateway, attack scenarios and CI are thin consumers of it. Private keys are PKCS#8 files with owner-only permissions or CI secrets, the index is SQLite, and the JSON envelopes on disk are the source of truth. The reference pipeline exports the scikit-learn digits dataset (8×8 images; 1 000 training, 437 fine-tuning, 360 held-out samples), attests the datasets with content checks (label distribution against a reference, duplicate rows, a known trigger pattern), trains a 38 282-parameter CNN, fine-tunes it and sends it through the gateway. A dashboard exposes the same operations; a Docker Compose stack and two GitHub Actions workflows are provided. *Reproducibility:* 168 automated tests run in 36 s; `attacks/run_all.py` regenerates every table, `scripts/benchmark.py` every cost figure, and `paper/figures/make_figures.py` writes every measured number of Section V to `analysis.json`. Code, tests and scenarios are public (https://github.com/domail2karandeshmukh-cyber/ai-chainguard), and a web page verifies the signed model of experiment E2 and runs it in the reader's browser (https://ai-chainguard.vercel.app).

V. Evaluation

A. Setup and protocol

Unless stated otherwise, numbers come from one run (seed 42) on 9 November 2026 in the reference development environment, a cloud Linux container with two logical Intel Xeon cores at 2.10 GHz, 8 GB of memory and no GPU (Python 3.13, PyTorch 2.15 on CPU). Each attack variant starts from a fresh copy of a clean, fully attested chain, applies one manipulation with the access the attacker would have, verifies the targeted artifact with its whole chain and records the outcome, the checks that fired and the wall time of that verification; a variant is correct when the observed outcome equals the predicted one, and untouched controls must stay VERIFIED. Backdoor experiments train real models: the poison is a BadNets-style 2×2 corner patch [13] applied to 8 % of the training rows with the label changed to 7, and the attack success rate (ASR) is the share of held-out images of other classes that, carrying the trigger, are classified as 7. The containerised pipeline, API and dashboard were also executed on a team laptop (MacBook Air) on 16 October 2026, and the tests and the release gate run on GitHub-hosted runners (Python 3.11) at every push; benchmarks were measured in the reference environment only.

B. Post-signing attacks are detected

Table III summarises the seven experiments; Fig. 3(a) shows every variant. All 48 variants ended with the predicted outcome. Every manipulation made after signing was rejected, including the subtle ones. On datasets: flipping the labels of five rows, appending one row or adding a single trailing space to a signed file (V1, V2, with the changed lines named). On models: nudging one weight by 10^{-6} or re-saving identical weights in another format (V1). On provenance: deleting a parent record, re-signing a modified parent with the legitimate key, editing a parent link without re-signing or presenting a model with no dataset ancestry (V5, V3, V8). On keys: corrupting or replaying signatures, using a key in the wrong role, or signing with unknown or revoked keys (V3, V4). Revoking an artifact or a key fails every descendant and blocks deployment, while a dataset signed by another role still verifies after a training-key revocation: the blast radius is the key's. Detection, measured as the verification time of the attacked artifact with its chain, took 1.7–15.1 ms (median 7.5 ms).

TABLE III
The attack suite: 48 variants, all with the predicted outcome. Checks: the ones that fired on attacks of that experiment.

Exp.	Experiment (scenario)	Var.	Outcome	Checks
1	Dataset tampering (A)	5	control VERIFIED; 4 FAILED	V1, V2, V6
2	Model substitution (B)	4	control VERIFIED; 3 FAILED	V1
3	Invalid signature, key misuse (D)	7	control VERIFIED; 6 FAILED	V3, V4, V6
4	Broken provenance (C)	5	control VERIFIED; 4 FAILED	V3, V5, V8
5	Revocation and 8 bypasses (F)	13	2 VERIFIED as predicted; 11 FAILED, gateway blocks	V3, V4, V6, V7
6	Poison before signing (E)	5	E0 control VERIFIED; E1 FAILED; E3 WARNING; E2, E4 VERIFIED (Fig. 2)	V6, V9
7	Untrusted external source (G)	9	3 VERIFIED; 4 FAILED; 2 REFUSED at import	V1, V3, V4, V6

C. Attacks on the trust itself

Experiment 5 (scenario F) targets the mechanisms that decide who is trusted: after a base model is revoked, eight bypasses, C1–C8, try to make its fine-tuned descendant verify again. Editing the index (deleting the revoked artifact's row, C1; deleting the revoked key, C2; overwriting the cached status, C3) already failed in that earlier version, because the verifier never reads a cached status and treats a missing index row as a revocation it cannot rule out, failing closed. Clearing the `revoked` flag (C4) was its documented limitation and now fails: the signed log still says revoked, and the disagreement raises an `INDEX_TAMPERING` event (V6, V7). A rogue key written into the index with the training role and used to re-sign a substituted model (C5) fails because the key never entered the signed log (V3, V4). Truncating the log to drop the revocation (C6), editing the record in place (C7) and deleting the log (C8) fail closed (V3, V4, V6, V7): the replay no longer ends at the pinned head or no longer verifies under the root key. What remains trusted is the keys directory: whoever holds the root key and can rewrite the pin controls the trust decisions.

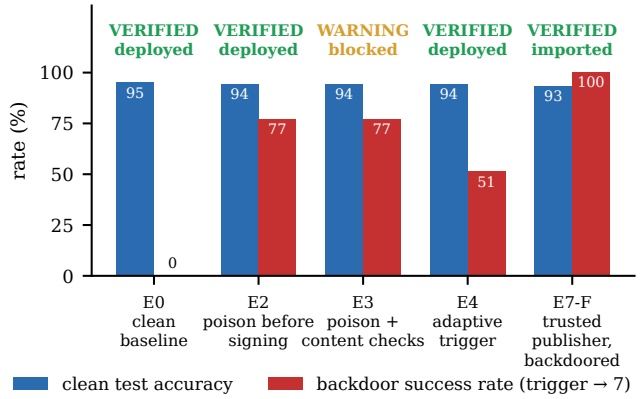


Fig. 2. Experiments 6 and 7: clean accuracy and backdoor success rate of the deployed or imported model, with its verification status. Signed poison (E2) and a trusted publisher's backdoor (E7-F) pass every cryptographic check; content checks catch a known trigger (E3) and miss an adaptive one (E4).

D. Signed is not safe

Experiments 6 and 7 (scenarios E and G) measure what the signature does not cover (Fig. 2; E0–E4 are the variants of Experiment 6). The clean pipeline (E0) deploys a model with 95.0 % clean accuracy and 0 % ASR. Poison injected into the signed dataset *after* signing (E1) fails the chain at V6, as expected. When the data steward signs a dataset poisoned *before* attestation (E2), every check passes: the fine-tuned model deploys as VERIFIED with 94.2 % clean accuracy, within one point of the clean model on ordinary inputs, and a 77.2 % ASR. With pre-attestation content checks (E3) the known corner trigger is flagged, the dataset carries a signed flag, the model verifies with WARNING and the gateway blocks it by default: heuristics work as evidence. An adaptive attacker who moves the trigger to a pattern the checks do not know (E4) deploys as VERIFIED with 51.2 % ASR. Two supplementary runs of 10 November confirm the boundary. Repeating E0, E2, E3 and E4 for ten other seeds changed no status: E2 deployed with a 74–95 % ASR (median 83 %) and E4 with 58–90 % (median 70 %), so the reference seed is at the low end. Poisoning the fine-tuning set instead (35 of its 437 rows) deploys as VERIFIED with a 25 % ASR, and gives WARNING with content checks. Experiment 7 repeats the lesson at the supply-chain boundary: a backdoored model trained by a *trusted*, registered publisher (variant E7-F) imports as VERIFIED with 93.3 % clean accuracy and a 100 % ASR, while the unknown publisher, the model modified in transit, the unsigned model and the revoked publisher key are all rejected (Table III). A valid signature proves *who* attested and *that nothing changed since*; it transfers trust to the signer and creates none.

E. Cost

Fig. 3(b) gives the cost. Attesting the four artifacts of the pipeline took 10.7 ms against 1 288 ms of training (0.83 %; workload-dependent, since attestation scales with artifact bytes and training with compute). A full-chain verification of the fine-tuned model with its three ancestors takes 8.5 ms (median of 200 runs; p95 12.2 ms); replaying the 5-record trust log accounts for 0.16 ms of it (0.75 ms cold). The first secure deployment costs 253 ms once: the chain verification with its audit-trail writes and the TOCTOU-safe read take 31 ms, the safetensors load 1.1 ms and the smoke test 8.3 ms; the remaining roughly 210 ms go to signing and registering the deployment record, which verifies the chain a second time, and to event and metric writes. A prediction then costs 0.084 ms before and 0.085 ms after the verified deployment: verification is not on the inference path. Signed envelopes are 1.4–3.1 KB per artifact; the row-level Merkle sidecar of a dataset is linear in its rows (30 and 68 KB here; one million rows hash in 1.7 s). SHA-256 runs at about 1.2 GB/s, so a 256 MiB model hashes in 220 ms; Ed25519 signing (0.04 ms) and verification (0.12–0.14 ms) do not depend on size. Replaying the trust log costs about 0.14 ms per record when every signature is verified (0.75, 6.9, 69 and 288 ms for 5, 50, 500 and 2 000 records) and 15 μs per record once signatures are cached within a process; with thousands of records one would checkpoint a signed snapshot rather than replay from genesis.

VI. Discussion and Limitations

The measurements are an upper bound on what the design offers. The attack suite is written by the defenders and runs on an 8×8-digit CNN: it shows that the cryptographic checks behave as designed and how large the semantic gap is on a small model, not that the gap has that size for a language model, where poisoning is cheaper [4]. The content checks are simple heuristics chosen to show that heuristics can be evaded (E4); stronger defences [21]–[24] would raise the bar but, as adaptive attacks show, not close it. Trust rests on long-lived Ed25519 keys in files: a stolen, unrevoked key signs valid manifests, which argues for short-lived keyless certificates [11] and hardware-backed keys. The trust log is local and single-writer, so a verifier holding a stale copy would miss a recent revocation, and the pin file is as critical as the root key. The canonical JSON is Python-only rather than RFC 8785 [26], the API has no authentication, the gateway verifies the chain twice on a first deployment, and benchmarks have run in one environment. None of this touches the central result, which holds by construction: signatures are evidence about origin and integrity, and a report that shows VERIFIED without stating what that status does and does not cover invites the wrong conclusion. AI-ChainGuard therefore keeps cryptographic and content evidence in separate checks (V0–V8 versus V9), labels external provenance that ends at a publisher, and never lets a heuristic produce a FAILED or a signature produce a "safe".

VII. Conclusion

We built an attestation chain for AI pipelines that links datasets, base models, external models, fine-tuned models and deployments through signed manifests that embed their parents' digests, enforces who may sign which stage, keeps trust and revocation in a signed, pinned, append-only log, and gates deployment on a full-chain verification. On 48 attack and control variants every manipulation made after signing

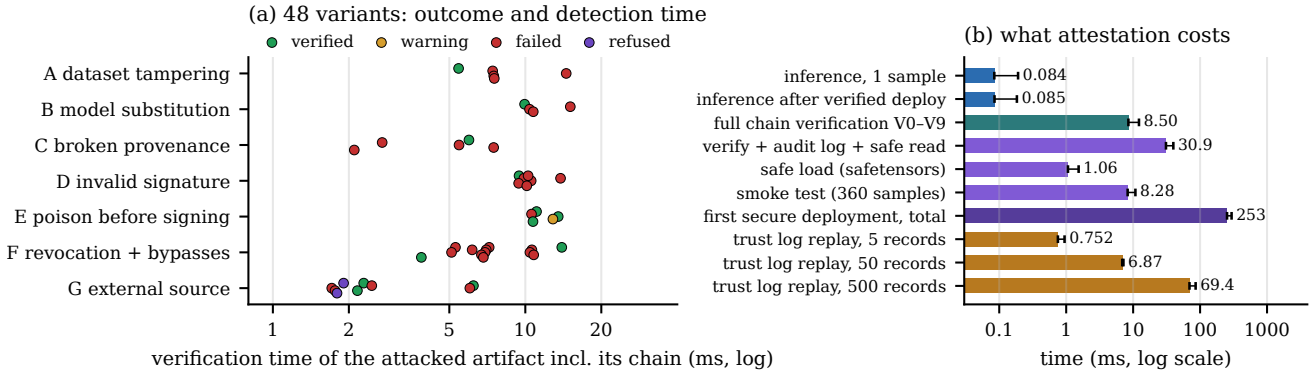


Fig. 3. (a) Outcome and verification time of all 48 variants, grouped by scenario (fresh chain per variant; the time includes the whole ancestry and the trust-log replay). (b) The cost of attestation in the reference environment (bar: median; whisker: p95): inference before and after a verified deployment, full-chain verification, the components of the first secure deployment, and the replay of trust logs of 5, 50 and 500 records with every signature verified.

was detected, including bypasses aimed at the index and at the log itself, at a cost of under one per cent of training time, 8.5 ms per verification and nothing per inference. The same suite shows the limit: poison present before signing verifies and deploys with a 77 % backdoor success rate, and a trusted publisher's backdoor with 100 %. AI supply-chain security needs both kinds of evidence, reported separately; this work provides the first kind and measures where the second must take over. Next steps are the publication of the trust log to a shared transparency log, keyless signing for the roles, reproduction on team hardware and with larger models.

Acknowledgment

The authors thank Sofiane Hamrioui, ESAIP Angers, for supervising this project. The implementation was developed with AI-assisted code generation and validated by execution; the team is responsible for its design, evaluation and conclusions.

References

[1] PyTorch team, "Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022," Dec. 31, 2022. [Online]. Available: https://pytorch.org/blog/compromised-nightly-dependency/. Accessed: Nov. 9, 2026.

[2] D. Cohen, JFrog Security Research, "Data scientists targeted by malicious Hugging Face ML models with silent backdoor," Feb. 2024. [Online]. Available: https://jfrog.com/blog/data-scientists-targeted-by-malicious-hugging-face-ml-models-with-silent-backdoor/. Accessed: Nov. 9, 2026.

[3] K. Zanki, ReversingLabs, "Malicious ML models discovered on Hugging Face platform," Feb. 2025. [Online]. Available: https://www.reversinglabs.com/blog/rl-identifies-malware-ml-model-hosted-on-hugging-face. Accessed: Nov. 9, 2026.

[4] A. Souly et al., "Poisoning attacks on LLMs require a near-constant number of poison samples," arXiv:2510.07192 [cs.LG], Oct. 2025.

[5] N. Carlini et al., "Poisoning web-scale training datasets is practical," in *Proc. IEEE Symp. Security and Privacy (SP)*, 2024, pp. 407–425.

[6] European Parliament and Council, "Regulation (EU) 2024/1689 of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," *Off. J. Eur. Union*, L, 2024/1689, Jul. 12, 2024, Art. 15(5).

[7] NIST, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023.

[8] MITRE, "ATLAS technique AML.T0010: AI supply chain compromise." [Online]. Available: https://atlas.mitre.org/techniques/AML.T0010. Accessed: Nov. 9, 2026.

[9] OpenSSF, "SLSA specification v1.1: Supply-chain Levels for Software Artifacts," Apr. 2025. [Online]. Available: https://slsa.dev/spec/v1.1/. Accessed: Nov. 9, 2026.

[10] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 1393–1410.

[11] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2022, pp. 2353–2367.

[12] OpenSSF, "Model Signing v1.0," Apr. 2025. [Online]. Available: https://github.com/sigstore/model-transparency. Accessed: Nov. 9, 2026.

[13] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," arXiv:1708.06733 [cs.CR], 2017.

[14] in-toto project, "in-toto Attestation Framework: Statement v1." [Online]. Available: https://github.com/in-toto/attestation. Accessed: Nov. 9, 2026.

[15] Secure Systems Lab, "DSSE: Dead Simple Signing Envelope, protocol v1." [Online]. Available: https://github.com/secure-systems-lab/dsse. Accessed: Nov. 9, 2026.

[16] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proc. ACM Conf. Computer and Communications Security (CCS)*, 2010, pp. 61–72.

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

[18] M. Mitchell et al., "Model cards for model reporting," in *Proc. Conf. Fairness, Accountability, and Transparency (FAT*)*, 2019, pp. 220–229.

[19] T. Gebru et al., "Datasheets for datasets," *Commun. ACM*, vol. 64, no. 12, pp. 86–92, 2021.

[20] Hugging Face, "Safetensors: a simple, safe way to store and distribute tensors," documentation; security audit by Trail of Bits, 2023. [Online]. Available: https://huggingface.co/docs/safetensors. Accessed: Nov. 9, 2026.

[21] B. Wang et al., "Neural Cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. IEEE Symp. Security and Privacy (SP)*, 2019, pp. 707–723.

[22] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A defence against trojan attacks on deep neural networks," in *Proc. Annu. Computer Security Applications Conf. (ACSAC)*, 2019, pp. 113–125.

[23] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[24] B. Chen et al., "Detecting backdoor attacks on deep neural networks by activation clustering," in *Proc. AAAI Workshop on Artificial Intelligence Safety (SafeAI)*, 2019.

[25] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, IETF, Jan. 2017.

[26] A. Rundgren, B. Jordan, and S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, IETF, Jun. 2020.