SBSeg 2026 Armação dos Búzios - RJ
VEJA OS SELOS DE ARTEFATOS

RAGtrap: Source Revocation and Indexed Provenance Lookup for Poisoned RAG Corpora

Cristhian Kapelinski[1], **Diego Kreutz**[1]

[1]AI Horizon Labs, Federal University of Pampa (UNIPAMPA)
cristhianavila.aluno@unipampa.edu.br, diegokreutz@unipampa.edu.br

Abstract. *Retrieval-augmented generation (RAG) answers questions from passages retrieved out of a corpus, and poisoned passages can steer those answers: in a published attack, 5 passages per target question sufficed to produce the attacker's chosen answer. Once a source is known to be compromised, recovery means finding the passages it supplied and removing them without discarding unrelated content.* RAGTRAP *records a signed provenance entry for every passage at ingestion, indexed by source and by content hash. Tracing a suspect passage is then one hash lookup and no language-model request, against one or more per passage for post-incident attribution in the literature. Revoking a source removes only its passages, whereas deleting whole documents also discards benign ones. Exact hashing cannot attribute content changed after ingestion, nor content supplied by more than one source, so* RAGTRAP *supports recovery from a known compromised source but does not detect or prevent poisoning.*

1. Introduction

A 2024 survey of 600 enterprise IT leaders found that 51% used retrieval-augmented generation (RAG) for generative-AI applications.[1] RAG retrieves short passages from a corpus and gives them to a large language model (LLM), which writes an answer from them [Huang and Huang 2026]. The corpus is divided into passage-sized fragments called *chunks*. Vector databases, which commonly store these chunks, grew 377% as a product category in one year.[2]

Retrieval followed by generation is also part of consumer web search. Google's AI Mode breaks a question into subtopics and issues many searches at once, a technique Google calls *query fan-out*.[3] Google reports that AI Mode has surpassed one billion monthly users.[4] A randomized field experiment found that Google AI Overviews reduced outbound organic clicks by 39.8%. They also increased searches with no outbound click by 34.5% [Agarwal and Sen 2026]. Publishers call the prospect of search referrals approaching zero "Google Zero."[5] These findings do not show that the open web will disappear. They show that retrieval-and-generation systems already operate at a scale that affects the websites supplying their content.

The same corpus creates a security risk because its passages may come from external websites and shared document repositories [Huang and Huang 2026, Zou et al. 2025]. A conventional ingestion pipeline records the text and its embedding, a numeric representation used for retrieval. It may not preserve a verifiable association between each chunk and the party that supplied it. An adversary who controls an ingested source can insert passages designed to influence later answers. PoisonedRAG demonstrates this attack by adding 5 malicious passages per target question to a knowledge base containing millions

1. Menlo Ventures, State of Generative AI in the Enterprise, 2024. 2. Databricks, State of AI: Enterprise Adoption, 2025. 3. Google, AI in Search: Going beyond information to intelligence, 2025. 4. Google, 100 things we announced at Google I/O 2026. 5. Axios, Publishers prepare for Google Zero, 2025.

of texts. The model returns the attacker's chosen answer for approximately 90% of the tested questions [Zou et al. 2025]. Thus, a few retrieved passages can affect an answer even when the overall corpus is large.

Related work shows that control over externally hosted data can be obtained at low cost. Carlini et al. studied two public image datasets whose entries refer to web addresses [Carlini et al. 2024]. Some addresses belonged to expired domains; registering those domains for US$60 would have allowed an attacker to control 0.01% of each dataset. The authors also demonstrate a second mechanism in which an attacker edits a Wikipedia page shortly before a dataset snapshot collects it. Although that study concerns training datasets rather than RAG, both settings include content hosted outside the system operator's control. The OWASP Top 10 for LLM Applications identifies data and model poisoning as a risk. It recommends tracking data origin, validating unverified sources, and maintaining dataset versions.[6]

After an operator identifies a compromised source, remediation requires two operations: mapping suspect chunks to the source that supplied them and removing all chunks associated with that source. Existing defenses address adjacent tasks. GMTP [Kim et al. 2025] is a query-time detector that drops suspicious passages from an answer's context, once per query; Section 2 gives its mechanism. It reduces the poisoned content the model sees, but neither records who supplied a passage nor removes anything from the stored corpus.

RAGForensics [Zhang et al. 2025] and RAGOrigin [Zhang et al. 2026] instead perform post-incident attribution, following earlier work on tracing poisoned training samples [Shan et al. 2022]. Given a suspicious answer and its retrieved passages, they use language-model scores to identify which passages likely caused the answer. This is a more general task because it does not assume provenance recorded during ingestion. It also requires model inference for each investigation and does not provide source-level revocation. A 2026 survey similarly reports limited coverage of integrity, provenance, knowledge-base governance, and rollback in RAG defenses [Xu et al. 2026].

We present RAGTRAP, a recovery layer implemented at the ingestion boundary. For each chunk, the gate stores a signed provenance record beside the corpus. The record identifies the source location and the party that supplied the content. We call the stable identifier for that party the *source identity*. Each record describes where the chunk came from: the address, written as a Uniform Resource Identifier (URI); the source identity; a SHA-256 hash of the chunk bytes; the detector verdicts; and an ingestion timestamp. The hash is a short fingerprint that changes if any byte changes. Appendix C shows a complete record. An Ed25519 signature lets anyone holding the public verification key check that these fields were not altered after admission.

Per-chunk signed provenance is an existing design choice rather than a contribution of this work. VectorSmuggle binds stored embeddings to hashes of their source text [Wanger 2026]; policy-governed RAG attaches verifiable receipts to answers [Ray 2025]; and MerkleSpeech stores chunk-level fingerprints in issuer-signed Merkle trees for audio [Ono 2026]. RAGTRAP uses this prior provenance mechanism to support two recovery operations for text RAG corpora: provenance lookup and source revocation.

Two indices support these operations, and Section 3 specifies both. A source index

6. OWASP, LLM04:2025 Data and Model Poisoning, 2025.

maps a source identity to its chunks, so revocation deletes what a given source supplied without touching unrelated chunks in the same documents. A content-hash index maps the SHA-256 hash of a suspect chunk to its provenance records, which resolves it in one hash-table lookup and no language-model request.

Two cases defeat that lookup: bytes that changed after admission, which we call *content drift*, and identical bytes admitted from more than one source. In both, the system returns no attribution rather than a guess, and Section 4 measures what the first costs. RAGTRAP is a recovery layer, not a detector: an external detector or analyst must first identify a suspect chunk or compromised source.

The paper asks one question: once a source is known to be compromised, how precisely and at what cost can an operator remove its content from a RAG corpus? We answer it with two mechanisms, both evaluated in Section 4. The first is *source revocation*. The source index lists the chunks a given source supplied, and the system deletes exactly those. We compare it with the usual alternative, deleting the whole document that contains a poisoned passage, and measure how many benign chunks each option removes and how long each takes. The second is *provenance lookup*. Given a suspect chunk, the content-hash index returns the source recorded for it at admission. We compare its cost with two published forensic procedures run on the same inputs, and measure how much of it survives when chunks change after ingestion.

The evaluation combines three third-party artifacts: the PoisonedRAG attack, the Natural Questions corpus, and retrieval results released by the RAGOrigin authors. The result is that recording provenance at ingestion turns recovery into an index operation. Removal touches no benign chunk, and attribution costs no language-model request. The price is failing on content that no longer matches the bytes that were admitted. These conclusions cover recovery from known compromised sources in this setting. They do not cover poisoning detection, integration with a production vector database, or attribution of text never seen at ingestion. The implementation and evaluation artifacts are publicly available (Section 5).

2. Related Work

Table 1 groups the approaches most directly related to recovery from corpus poisoning into four families: the attack itself, query-time filtering, post-incident attribution, and admission-time attestation. Five dimensions separate them: the pipeline stage where each mechanism runs, the provenance it records, the smallest unit it can address, the evidence it uses for traceback, and whether it describes source revocation. General cryptographic and retrieval building blocks remain in prose because they do not perform this recovery task directly.

Detection and query-time filtering. GMTP runs on the top-k passages a query returns, before those passages reach the generator [Kim et al. 2025]. For each one, it identifies the terms that most affect the retrieval score, masks them, and uses a masked language model to estimate how probable they are in context. A low probability indicates that the passage may have been constructed to manipulate retrieval. GMTP excludes passages classified as poisoned and reports filtering more than 90% of poisoned content during retrieval. RevPRAG detects poisoned responses from internal activations of the generating model [Tan et al. 2025]. RAGPart partitions documents before retrieval, whereas RAGMask flags terms whose removal substantially changes retrieval similarity [Pathmanathan et al. 2025]. These methods reduce the probability that poisoned text

Table 1. Comparable approaches versus RAGTRAP, grouped by recovery task. A mark records what the published design describes, not what it could be extended to do. "–": not provided; *: attack.

Work	Acts at	Provenance recorded	Smallest unit	Traceback evidence	Source revocation
Corpus-poisoning attack					
PoisonedRAG* [Zou et al. 2025]	Ingestion	–	Chunk	–	×
Query-time filtering					
GMTP [Kim et al. 2025]	Retrieval	–	Chunk	–	×
Post-incident attribution					
RAGForensics [Zhang et al. 2025]	Post-incident	–	Chunk	LLM classification	×
RAGOrigin [Zhang et al. 2026]	Post-incident	–	Chunk	Proxy-model scores	×
Admission-time attestation and provenance					
Cosign [Newman et al. 2022]	Ingestion	Keyless signature	Artifact	Signer certificate	×
D-RAG [Andersen et al. 2025]	Ingestion + retrieval	On-chain hashes	Document	–	×
VectorSmuggle [Wanger 2026]	Ingestion	Ed25519 signature	Embedding	–	×
MerkleSpeech [Ono 2026]	Ingestion	Merkle root + issuer sig.	Audio chunk	Merkle proof	×
RAGShield [Patil 2026]	Ingestion + retrieval	Signed attestation	Document	Attestation record	×
RAGTRAP (this work)	**Ingestion + post-incident**	**Ed25519 signature**	**Chunk**	**SHA-256 content hash**	✓

reaches answer generation. Their output applies to the current query and does not remove the source's other content from the corpus. RAGTRAP addresses this separate recovery operation and therefore complements query-time detection.

Forensic attribution. Poison Forensics established pipeline-level traceback, trimming innocent training samples by iterative clustering and unlearning until only the samples responsible for the attack remain [Shan et al. 2022]. Two RAG-specific attributors build on that lineage. RAGForensics re-retrieves the candidate texts behind a reported bad answer. A language model then judges each one poisoned or benign. It reports detection accuracy above 97% [Zhang et al. 2025]. RAGOrigin extends the idea to black-box responsibility attribution [Zhang et al. 2026]. It scores each candidate on retrieval rank, semantic relevance, and its influence on the generated answer, then separates poisoned from benign by clustering those scores.

Both procedures attribute suspicious passages without requiring provenance recorded at ingestion, and both can delete the passages they identify. Their published workflows do not associate those passages with a supplying source or revoke the source's remaining content. Because they infer responsibility during each investigation, they require language-model requests for the evaluated candidates. When signed provenance is available, RAGTRAP reads the recorded source instead; when it is unavailable, forensic inference remains necessary. Section 4 runs both procedures on the same suspect passages used to evaluate RAGTRAP.

Ingestion-time attestation. Sigstore/Cosign binds a signed artifact to a publisher identity and records the event in a transparency log [Newman et al. 2022]. Its unit is a complete artifact rather than a retrieved text fragment. D-RAG admits a document and its sources to a decentralized RAG database after a privacy-preserving community vote. It records their hashes on a blockchain, and at retrieval it records a hash of the generated response so the user can detect tampering [Andersen et al. 2025]. Both establish

admission-time evidence, but neither provides per-chunk traceback followed by source revocation.

Cryptographic provenance and receipts. VectorSmuggle signs hashes of stored embeddings and their source text to detect embedding-store modification [Wanger 2026]. Policy-governed RAG binds answers to signed evidence receipts that auditors can verify offline [Ray 2025], while ZKPROV proves that a response used a certified dataset [Namazi et al. 2025]. MerkleSpeech is the closest design in structure: it stores audio-chunk fingerprints in a signed Merkle tree and verifies individual chunks through inclusion proofs [Ono 2026]. These systems establish provenance at different granularities, but do not combine exact content-hash lookup with source-level removal in a text RAG corpus. RAGTRAP uses signed provenance as an existing building block and contributes these recovery operations.

Provenance defense-in-depth. RAGShield attaches signed attestations to documents, weights retrieval by source trust, and flags cross-source contradictions [Patil 2026]. It reports 0.0% attack success across five adversary tiers. For an insider who edits a document in place, it reports 17.5%.[7] Its source identifier supports attribution, but its unit is a complete document. The paper does not describe removal of all content from a compromised source. We compare capabilities because no artifact is available.

Among the approaches in Table 1, RAGTRAP is the only one whose published design combines per-chunk provenance, exact-hash source lookup, and removal of all chunks associated with a source. The crosses record scope rather than impossibility. MerkleSpeech and RAGShield do carry a source identity, and Sigstore has strong revocation for its own service keys, but none of the three describes removing a source's already-stored content. This positioning is limited to the works and dimensions compared.

3. Architecture

3.1. Terms used throughout

RAG pipelines have their own vocabulary. We fix it here, once, so the rest of the paper can use it without interruption.

- **Chunk.** Ingestion splits a long document into passage-sized fragments. These fragments, and not whole documents, are what the retriever ranks and what the model reads. RAGTRAP records provenance at this granularity.
- **Ingestion.** The one-time path that admits content into the corpus: load a document, split it into chunks, embed each chunk, and index it. It runs before any question is asked.
- **Source identity.** The stable name of the party that supplied the content, as opposed to the address where the content happened to sit. Two pages on one compromised domain share a source identity even though their URIs differ. The artifact names this field `principal`.
- **Traceback.** Answering the question "which source supplied this chunk?" for one suspect chunk.
- **Revocation.** Deleting every chunk recorded under one source identity, and refusing later content from it.
- **False purge.** A benign chunk deleted as collateral damage while removing poisoned ones. The *false-purge rate* is the fraction of deleted chunks that were benign.

7. We cite the v1 preprint (1 April 2026); the current arXiv version addresses a different problem.

- **Content drift.** Any difference between the suspect text an investigator holds and the bytes that were admitted. Reading the suspect back from the store avoids it; it arises when the suspect arrives by another route, such as an answer log or a stage that rewrites text after retrieval.

3.2. What the gate records at ingestion

Figure 1 shows the two halves of the system. The left half runs once per chunk, at ingestion. The right half runs only after an incident.

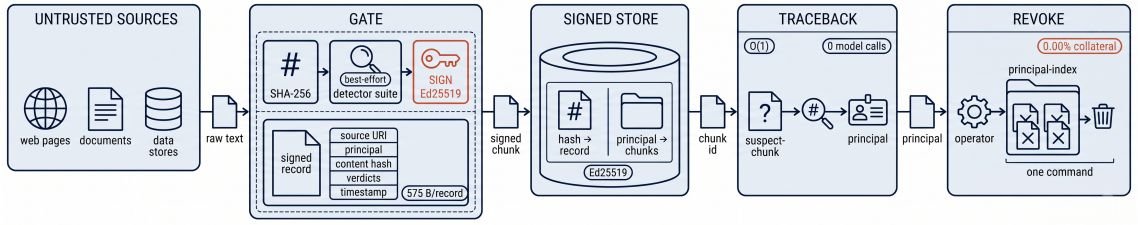


Figure 1. RAGTRAP ingestion, per-suspect provenance lookup, and source revocation.

Reading the left half of the figure from the untrusted source inward: the document loader divides the source into chunks; the gate fingerprints and inspects each chunk; the gate signs a record describing it; and the chunk, its record, and two index entries land in the store. Algorithm 1 in Appendix A states the same path precisely.

Two properties of Algorithm 1 matter later. First, the hash covers the chunk's exact UTF-8 bytes, with no normalization. That exactness is what lets a lookup answer without a model request. It is also the source of the drift limitation measured in Section 4. Second, the signature covers the detector verdicts and the content hash together, so no field can be altered after admission without breaking it, and anyone holding the public verification key can check that offline. It does not by itself bind a record to a chunk: a record reattached to different bytes still verifies. Recomputing the hash is what catches that, and lookup does exactly this. The prototype records detector verdicts without enforcing admission; production should quarantine suspect chunks. The signature certifies *where a chunk came from*, not that it is safe.

3.3. How recovery uses the two indices

The store keeps two side indices, one for each recovery question. The content-hash index I_H maps a fingerprint to the chunks carrying it, and answers "who supplied this text?". The source index I_S maps a source identity to its chunks, and answers "what else did this source supply?". Algorithm 2 in Appendix A gives both procedures.

TRACEBACK returns a source only when every record holding those exact bytes names the same one. When the bytes were never admitted, and when two different sources supplied byte-identical text, it returns $\perp$: no attribution, rather than a guess. REVOKE reads its target set straight from I_S, so its cost depends on how much the compromised source supplied, not on how large the corpus is. Both operations are index reads; neither consults a language model. Appendix B walks one incident through both end to end.

Deployment contract. The gate sits between the document loader and the vector database. A deployment should log the chunk identifiers it retrieves for each answer, and an investigation should read those chunks back from the store. Lookup is then exact by construction, since the bytes hashed are the bytes signed. On revocation, the host application deletes the indexed chunk rows, their embeddings, and any caches keyed by

those chunk IDs. A document update creates new chunk hashes and records, then retires the prior IDs. Cache invalidation and version retention remain responsibilities of the host RAG platform, because platform interfaces vary. The signed metadata and the two indices provide the IDs that each of those operations must update. The prototype stores chunks and records in an in-memory datastore; a deployment would persist them beside the embeddings.

Threat model and scope. The attacker controls or compromises a web source ingested by the corpus and injects poisoned texts. This includes split-view and frontrunning attacks, in which a source changes after collection [Zou et al. 2025, Carlini et al. 2024]. The protected asset is the *traceability* of the indexed corpus and the integrity of the provenance records. The operator is trusted, holds the private key, and seals provenance at ingestion; editing stored text in place therefore falls outside this model, since it needs that same write access. RAGTRAP provides forensic and operational support, while poisoning detection remains outside its scope and query-time defenses remain complementary [Kim et al. 2025, Pathmanathan et al. 2025].

4. Evaluation

Three experiments answer three questions. **Exp. 1** compares attribution cost and measures the effect of content drift. **Exp. 2** compares collateral removal and in-memory removal latency with document-level purging. **Exp. 3** tests whether the suspect passages change generated answers. A separate instrument check, reported in Appendix F, confirms on a corpus whose ground truth we control that every record verifies and that a tampered message is rejected. Appendix E tabulates each experiment with its dataset, parameters, and reported metric (Table 5). Attack data, retrieval results, labels, and baseline procedures come from third-party artifacts. We measure latency, throughput, and speedup on one host: an AMD Ryzen 5 8600G with 6 cores and 12 threads, 32 GB RAM, and an NVIDIA RTX 5060 Ti with 16 GB. RAGTRAP's own operations are single-threaded CPU work and use no GPU, while both baselines run their language model on the GPU in half precision. The comparison therefore gives the baselines the faster hardware. We measured RAGTRAP's attribution latency 20 times and report the fastest run; the average was 1.3% slower. Each baseline was timed once, and the scaling points use 2 to 5 repeats as the corpus grows. Measurements vary by up to $\sim$20% across runs.

Three datasets form one evaluation scenario rather than three independent ones. BEIR Natural Questions supplies the clean Wikipedia corpus. PoisonedRAG supplies adversarial passages for its questions. The RAGOrigin feedback file supplies candidates retrieved by `e5`, together with third-party poison and clean labels. This combination lets all attribution procedures use the same questions, passages, and labels. Table 6 in Appendix E gives sizes and licenses.

Exp. 1: Forensic-time attribution on the real attack. For each of the 100 questions, the feedback file lists the passages returned by `e5`, a dense retriever that represents questions and passages as numeric vectors and ranks them by similarity [Wang et al. 2022], and labels each passage poisoned or clean. The released data carry no admission provenance. We therefore assign one attacker identity to labeled poisoned passages and synthetic benign identities to the remaining passages. We then ingest and sign every chunk with Ed25519. The file holds more passages than a deployment normally sends to the model, so we use the top-10 per question, producing 1,000 suspects: 500 poisoned and 500 clean. Exp. 3 separately evaluates the top-5 passages sent to the generator.

We run three attribution procedures on the same 1,000 suspects. The first is RAGTRAP's content-hash lookup, one per suspect. The second is RAGForensics, which requests one poisoned-or-benign classification per passage from a local Qwen2.5-3B-Instruct model. It has about 3 billion parameters, so it needs roughly 6 GB in half precision and fits the reference GPU. The third is RAGOrigin, which makes two scoring requests per passage to the same local model.

Table 2. Exp. 1 attribution cost on identical 1,000 real suspects.

Cost metric	RAGTRAP	RAGForensics	RAGOrigin
Latency / suspect	**78.7 μs**	1.7 s	64.5 ms
Model requests	**0**	1,000	2,000
Cost vs. lookup (×)	**1**	21,026	819

Table 2 compares the measured computational cost of the three procedures. RAGTRAP retrieves the recorded source in 78.7 μs per suspect without a model request. RAGForensics takes 1.7 s for one classification request. RAGOrigin takes 64.5 ms for two scoring requests. On the reference machine, these latencies are 21,026× and 819× the lookup latency, respectively. The procedures assume different available information. The baselines infer responsibility from text, whereas RAGTRAP retrieves provenance stored at ingestion (Section 3).

Both baselines identify poisoned passages accurately without any ingestion-time provenance. Table 4 in Appendix D reports their recall, precision, and false-positive rate with 95% confidence intervals (CIs). We do not place RAGTRAP in that table, because a direct accuracy comparison would conflate two different tasks. The baselines classify a passage from its text. RAGTRAP returns a source that was recorded at admission, so its failures are of a different kind: byte mismatches and content supplied by more than one source. Those are what we evaluate next.

Drift sensitivity: recall when the suspect is not the stored bytes. Read back from the store, a suspect cannot miss. This experiment measures the other case, in which the investigator holds text close to an admitted chunk but not byte-identical to it, having recovered it from an answer log or from a stage that reformats after retrieval. We model that by perturbing the suspect text and leaving the store untouched. RAGTRAP resolves a suspect only when its bytes match admitted content and the matching records agree on one source. Without drift, every suspect's bytes are in the index by construction, so the only way to miss one is disagreement between sources. Exactly five poisoned suspects fall in that case: their bytes are also recorded under a benign source identity, and the lookup returns no attribution rather than picking one. We independently rewrite each of the 500 poisoned suspects with probability p, using seed 1337. Recall is the fraction of poisoned chunks traced to one source. It is 0.99 at $p{=}0$, 0.69 at $p{=}0.3$, and 0.50 at $p{=}0.5$ (Figure 2a). Rewritten chunks add hash misses to those five cross-source cases. We report Wilson 95% CIs: $[0.65, 0.73]$ and $[0.46, 0.55]$. The corresponding standard error is about two percentage points, so another random split would typically move recall by that much. Precision remains 1.00 because the lookup returns a source only for unambiguous exact matches.

What decides the outcome is not how the attacker behaves but how the investigator obtained the suspect text, and Table 8 in Appendix H sorts the cases on that axis. Reusing stored bytes always resolves. Re-deriving the text fails, whether it was copied out of a log, rewritten by a stage running after retrieval, or collected again from the source. Most of these have procedural answers rather than cryptographic ones: log chunk

identifiers, hash before any post-retrieval rewriting, and trace the stored chunk instead of the live page. Two cases remain genuinely open. Content that never passed the gate has no record to find, and identical bytes admitted from two sources cannot be separated by content at all. Tolerating re-derived text would need near-duplicate or semantic matching such as MinHash [Broder 1997], SimHash [Charikar 2002], or locality-sensitive hashing [Indyk and Motwani 1998], which is future work (Section 5).

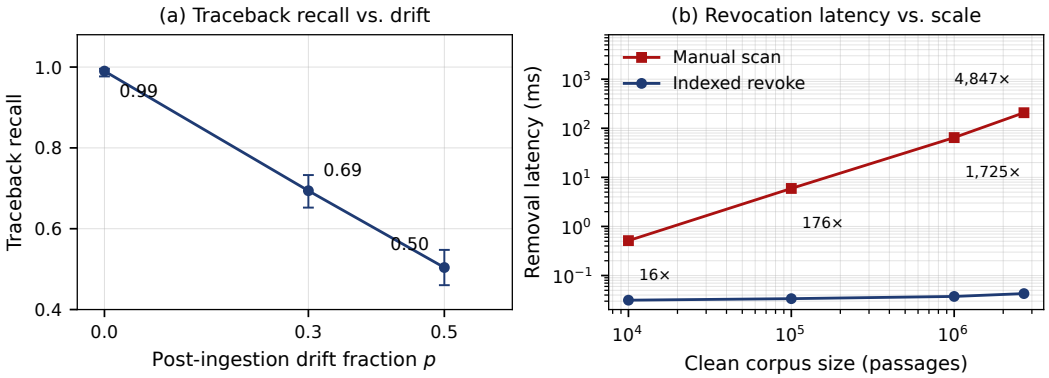


Figure 2. Results: (a) traceback recall vs. drift fraction p (95% Wilson CIs); (b) Exp. 2 indexed removal latency vs. manual scan across corpus scale (log axes; manual/revoke ratio annotated).

Exp. 2: Revocation precision and cost at scale. We create 206 mixed documents. Their benign NQ chunks retain a benign source identity, while injected PoisonedRAG passages share one attacker identity. With 3 injected passages per document, document-level purging removes 1,290 chunks. Of these, 672 are clean. The false-purge rate, the fraction of removed chunks that are benign, is therefore 0.52 (95% CI [0.49, 0.55]). Source-indexed revocation calls the source index and removes 618 attacker-supplied chunks and no benign-source chunks. Poison labels are used only to measure the result, not to select chunks for removal. Every document here carries at least one injected passage, so document-level purging deletes all 206 of them at every density: the benign loss is the same 672 chunks, the entire benign content, in all four configurations, and only the ratio moves with the poison count. The false-purge rate rises from 0.39 at five injected passages per document to 0.77 at one (Table 7, Appendix G). Source-indexed revocation leaves all 672 benign chunks in place, which is its main benefit.

We measure in-memory removal latency by treating one PoisonedRAG source as compromised. We locate its chunks with and without the source index. Ingestion divides 2,681,468 clean source passages into chunks and adds 500 PoisonedRAG chunks, for 4,364,162 chunks in total. The index lookup has expected $O(1)$ time, while deleting m chunk identifiers costs $O(m)$. On this full corpus, locating and deleting $m = 100$ chunks takes 42.8 μs. Scanning the in-memory corpus for the same chunks takes 208 ms. Figure 2b shows that the measured ratio grows from 16$\times$ at 10,000 passages to 4,847$\times$ on the full corpus. The scan grows with corpus size, whereas the indexed operation depends primarily on the number of revoked chunks. These times exclude vector-database persistence, network access, cache invalidation, and detector latency.

The index also adds ingestion and storage costs. Each Ed25519 signature takes approximately 55.7 μs, or 17,961 chunks/s on one thread. The signature occupies 64 bytes within a 575-byte record. Signing is 1.9$\times$ slower than the symmetric HMAC reference, which takes 29.0 μs and produces a 32-byte tag. Unlike HMAC, Ed25519

permits verification without sharing the signing key. Provenance records require 575 MB per million chunks and 57.5 GB per 100 million chunks, excluding embeddings and database indices.

Exp. 3: Effect on generated answers. We provide the top-5 retrieved passages to a local Qwen2.5-3B-Instruct generation model. We then compare its answer with the attacker's target. The answer matches that target for 98% of the 100 questions (95% CI [93, 99]). PoisonedRAG reports 97% on the same corpus with a different generator [Zou et al. 2025]. This reproduction confirms that the suspect passages affect downstream answers under our setup. We do not present it as a new attack result.

5. Limitations and Conclusion

Scope: what RAGTRAP is and is not. The recovery layer starts after detection. An external detector or analyst must first flag a suspect chunk or compromised source. Until then, RAGTRAP takes no recovery action. Detection remains outside our scope, and query-time defenses are complementary [Kim et al. 2025, Pathmanathan et al. 2025]. Attribution requires an exact, source-unambiguous content-hash match. Reading the suspect back from the store meets that requirement by construction, so the recall loss we report applies only to investigations working from text obtained some other way (Table 8). A split view changes what a source serves later without altering the bytes already stored, so it costs re-collection rather than lookup recall. Identical bytes from different sources stay unattributed. Future work should add near-duplicate or semantic matching so traceback tolerates content drift. Candidate methods include MinHash, SimHash, and locality-sensitive hashing [Broder 1997, Charikar 2002, Indyk and Motwani 1998].

Threats to validity. Exp. 1 uses one released attack-feedback set over NQ. Its source identities are assigned from third-party labels rather than recorded by the original datasets. The scaling half of Exp. 2 uses the full 2,681,468-passage corpus, although lookup complexity and indexed revocation do not depend on this scale. The prototype uses an in-memory datastore, so its latency results do not estimate end-to-end remediation time in a deployed vector database. We run RAGForensics and RAGOrigin with local open models rather than the hosted models used in their papers, and decode the judge greedily rather than with the sampling settings of its released code. Future evaluations should cover adaptive and multiple attackers on corpora such as HotpotQA and MS-MARCO. A future implementation should also use threshold or multi-party key custody to reduce dependence on one trusted key holder. The present design assumes a trusted operator, so private-key compromise remains a risk.

Conclusion. Published attacks show that a few poisoned passages can influence answers produced from a much larger RAG corpus [Zou et al. 2025], and existing filters and post-incident attributors do not remove all content associated with a known compromised source. RAGTRAP records signed provenance at ingestion and uses two indices for recovery. In our evaluation, source-indexed revocation removed no clean chunk where document-level removal had a 0.52 false-purge rate, and indexed lookup traced 0.99 of poisoned suspects without a single model request. What it cannot match exactly it leaves unattributed rather than guessed. RAGTRAP is open.[8]

8. https://doi.org/10.5281/zenodo.22664942. This work was partially supported by Capes, CNPq (Grant No. 409743/2025-9), and FAPERGS (Grant Nos. 24/2551-0001368-7 and 24/2551-0000726-1). We thank the anonymous reviewers of SBSeg 2026. We also acknowledge using Generative AI to review and improve the code and manuscript. Google Gemini generated the schematic illustration in Figure 1.

References

Agarwal, S. and Sen, A. (2026). The impact of Google AI Overviews on publisher traffic and user experience: Evidence from a field experiment. *SSRN Working Paper 6513059*.

Andersen, T. E., Avalos, A. M., Dagher, G. G., and Long, M. (2025). D-RAG: A privacy-preserving framework for decentralized RAG using blockchain. In *Computer Science & Information Technology (CS & IT)*, pages 183–198.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*.

Carlini, N. et al. (2024). Poisoning web-scale training datasets is practical. In *IEEE S&P*.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *STOC*.

Huang, Y. and Huang, J. X. (2026). A survey on retrieval-augmented text generation for large language models. *ACM Computing Surveys*, 58(12).

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*.

Kim, S., Kim, J., Jeon, Y., and Lee, G. G. (2025). Safeguarding RAG pipelines with GMTP: A gradient-based masked token probability method for poisoned document detection. In *ACL Findings*.

Kwiatkowski, T. et al. (2019). Natural Questions: A benchmark for question answering research. *TACL*, 7.

Namazi, M., Nemecek, A., and Ayday, E. (2025). ZKPROV: A zero-knowledge approach to dataset provenance for large language models. *arXiv:2506.20915*.

Newman, Z., Meyers, J. S., and Torres-Arias, S. (2022). Sigstore: Software signing for everybody. In *ACM CCS*, pages 2353–2367.

Ono, T. (2026). MerkleSpeech: Public-key verifiable, chunk-localised speech provenance via perceptual fingerprints and merkle commitments. *arXiv:2602.10166*.

Pathmanathan, P., Panaitescu-Liess, M.-A., Chiang, C.-Y. J., and Huang, F. (2025). RAGPart & RAGMask: Retrieval-stage defenses against corpus poisoning in retrieval-augmented generation. *arXiv:2512.24268*.

Patil, K. (2026). RAGShield: Provenance-verified defense-in-depth against knowledge base poisoning in government retrieval-augmented generation systems. *arXiv:2604.00387v1*. Cited v1 (1 Apr 2026); v2 title and claims differ.

Ray, J.-M. L. (2025). Policy-governed RAG: Research design study. *arXiv:2510.19877*.

Shan, S., Bhagoji, A. N., Zheng, H., and Zhao, B. Y. (2022). Poison forensics: Traceback of data poisoning attacks in neural networks. In *USENIX Security*.

Tan, X. et al. (2025). RevPRAG: Revealing poisoning attacks in retrieval-augmented generation through LLM activation analysis. In *EMNLP Findings*.

Thakur, N. et al. (2021). BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS Datasets and Benchmarks*.

Wang, L. et al. (2022). Text embeddings by weakly-supervised contrastive pre-training. *arXiv:2212.03533*.

Wanger, J. (2026). VectorSmuggle: Steganographic exfiltration in embedding stores and a cryptographic provenance defense. *arXiv:2605.13764*.

Xu, Y. et al. (2026). Securing retrieval-augmented generation: A taxonomy of attacks, defenses, and future directions. *arXiv:2604.08304*.

Zhang, B. et al. (2025). Traceback of poisoning attacks to retrieval-augmented generation. In *WWW*.

Zhang, B. et al. (2026). Who taught the lie? responsibility attribution for poisoned knowledge in retrieval-augmented generation. In *IEEE S&P*. arXiv:2509.13772.

Zou, W., Geng, R., Wang, B., and Jia, J. (2025). PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *USENIX Security*.

A. Algorithms

Algorithm 1 is the ingestion path described in Section 3; Algorithm 2 is the pair of recovery procedures. Both use these symbols. c is a chunk's text and id its identifier; u is the URI it came from and s the source identity recorded for it; h is the SHA-256 hash of c and v the detector verdicts. r is one provenance record, σ its signature under the signing key sk, and $r.s$, $r.h$ its source and hash fields. The store $\mathcal{D}$ maps an identifier to its text, record, and signature; the indices map the other way, I_H from a hash to the identifiers carrying it and I_S from a source identity to the identifiers it supplied. Inside the procedures R is the set of records sharing a queried hash, T the identifiers to revoke, and $\perp$ no attribution.

Algorithm 1: Seal one chunk at ingestion

Input: chunk text c, source URI u, source identity s, chunk id id, signing key sk
Output: updated store $\mathcal{D}$, content-hash index I_H, source index I_S

$h \leftarrow \mathrm{SHA256}(\mathrm{UTF8}(c))$; `// fingerprint of the exact bytes`
$v \leftarrow$ RUNDETECTORS(c); `// verdict recorded; policy is deployment-specific`
$r \leftarrow \langle id,\ u,\ s,\ h,\ v,\ \textsc{Now}(),\ \textsc{SignerId}(sk),\ \text{“chunk”}\rangle$;
$\sigma \leftarrow \textsc{Sign}_{sk}(\textsc{Canonical}(r))$; `// key-sorted compact JSON`
$\mathcal{D}[id] \leftarrow (c,\ r,\ \sigma)$;
$I_H[h] \leftarrow I_H[h] \cup \{id\}$; `// content-hash index`
$I_S[s] \leftarrow I_S[s] \cup \{id\}$;

Algorithm 2: Recovery: traceback of one suspect, and revocation of one source

procedure TRACEBACK*(suspect text c)*:
  $h \leftarrow \mathrm{SHA256}(\mathrm{UTF8}(c))$;
  $R \leftarrow \{\, r : id \in I_H[h],\ (\cdot, r, \cdot) = \mathcal{D}[id] \,\}$; `// one hash-table probe`
  if $R = \emptyset$ ***or*** $|\{\, r.s : r \in R \,\}| \neq 1$ **then return** $\perp$;
  pick any $r \in R$;
  if *not* $\textsc{Verify}(r, \sigma_r)$ **then return** $\perp$;
  return $r.s$; `// recorded source; no model request`
procedure REVOKE*(source identity s)*:
  $T \leftarrow I_S[s]$; `// no corpus scan`
  foreach $id \in T$ **do**
    $(\cdot, r, \cdot) \leftarrow \mathcal{D}[id]$;
    delete $\mathcal{D}[id]$; $I_H[r.h] \leftarrow I_H[r.h] \setminus \{id\}$; $I_S[s] \leftarrow I_S[s] \setminus \{id\}$;
  mark s revoked; `// later content from s is refused`
  return $|T|$;

B. A worked incident

A worked incident. Suppose an analyst reports one bad answer and the query-time filter hands over the passage that produced it. TRACEBACK hashes that passage and probes I_H once, in 78.7 μs on our reference machine, and returns the source recorded when the passage was admitted. The operator confirms the source is compromised. REVOKE then reads that source's chunk set from I_S and deletes it. On the 4,364,162-chunk corpus of the scaling measurement, that is 100 chunks in 42.8 μs, with no benign chunk touched.

The alternative without provenance is to delete every document that contained a poisoned passage. In the 206-document corpus of Exp. 2, where each document holds one injected passage, that deletes all 206 documents: 878 chunks, 672 of them benign.

C. A provenance record

Below is a record the prototype produced for one chunk, shown with its fields sorted. The signature covers exactly these fields minus `signature_hex`, serialized as key-sorted compact JSON, so a verifier reconstructs the signed bytes from the record itself. `content_hash` is the SHA-256 of the chunk's UTF-8 bytes; `principal` is the source identity that both indices are keyed on.

Table 3. A signed provenance record for one chunk (hashes and keys truncated for width).

Field	Value
chunk_id	doc-42::c3
source_uri	https://example.org/paris
principal	example.org
content_hash	82604b5ba87febf461c893913fabaffe6168e42...
detector_verdicts	{"entropy": "benign", "repetition": "benign"}
timestamp	2026-08-04T05:38:12.383220+00:00
granularity	chunk
signer_name	ed25519
signer_identity	ed25519:f4f64d2505953c50cb4d7a5fa8bbc60f...
signature_hex	f90d7ba9fd313e0550deab49cb66f54337ea42...

D. Baseline attribution accuracy

Table 4 reports how accurately the two forensic baselines separate poisoned from clean passages on the 1,000 suspects of Section 4. Both do so from the passage text alone, without any provenance recorded at ingestion, which is the setting their papers target. We report these figures for completeness and do not place RAGTRAP beside them: it answers a different question, returning a source recorded at admission rather than classifying a passage.

Table 4. Baseline attribution accuracy on the 1,000 real suspects (95% CIs).

Procedure	Recall	Precision	False-positive rate
RAGForensics	0.96 [0.93, 0.97]	0.88 [0.85, 0.91]	0.13
RAGOrigin	1.00 [0.99, 1.00]	0.99 [0.97, 0.99]	0.01

E. Experimental setup

Table 5 states what each experiment exercises and the metric it reports; Table 6 gives the size and license of each dataset. All three are third-party artifacts, unchanged.

Table 5. Experiment design: what each test exercises and the metric reported.

Exp.	Question	Dataset	Parameters	Metric
Check	Do records verify and reject tampering?	Labeled corpus	200 chunks, 1 attacker source	Verify, tamper-reject, collateral
1	What are attribution cost and drift sensitivity?	RAGOrigin feedback over Natural Questions	100 questions, top-10, drift $p \in \{0, .3, .5\}$	Cost (latency, calls); recall
2	How precise and costly is revocation?	BEIR `nq` + PoisonedRAG	up to 2,681,468 passages, 206 mixed docs	False-purge rate, removal and sign latency
3	Do suspects change generated answers?	RAGOrigin feedback over Natural Questions	100 questions, top-5	Attack success

Table 6. Datasets.

Dataset	Contents	Size	License
BEIR `nq` [Thakur et al. 2021, Kwiatkowski et al. 2019]	Wikipedia passages, our clean substrate	2,681,468 passages	CC BY-SA 3.0
PoisonedRAG `nq` [Zou et al. 2025]	Adversarial passages, 100 questions, 5 each	500 passages	MIT
RAGOrigin feedback [Zhang et al. 2026]	`e5`-retrieved contexts per question, poison/clean labels	100 questions; top-10 of each taken as suspects (1,000)	MIT

F. Instrument validation

Before the real-data comparison, we check that the implementation does what the design claims on a corpus whose ground truth we control, using a seeded synthetic corpus of 200 chunks from five sources. One attacker source supplies 20 adversarial chunks. We ingest and sign every chunk, verify each record, and replace one signed message to test tamper rejection. We then revoke the attacker source and compare the removed identifiers with its known chunk set. All records verify, and the modified message is rejected. Revocation removes the attacker's 20 chunks without removing chunks assigned to other sources. These tests check deterministic implementation properties before the real-data comparison.

G. Document-level false purge by injection density

Table 7 reports the sweep of Section 4. The benign column is constant because every document holds poison at every density, so document-level purging always deletes all 206.

Table 7. Document-level false-purge rate by injected passages per document.

Injected / doc	Chunks purged	False-purge rate	95% CI
5	1,702	**0.39**	[0.37, 0.42]
3	1,290	**0.52**	[0.49, 0.55]
2	1,084	**0.62**	[0.59, 0.65]
1	878	**0.77**	[0.74, 0.79]

H. When exact-hash attribution holds

Table 8 sorts the cases of Section 4 by how the investigator obtained the suspect text, which is what decides whether the lookup resolves.

Table 8. When exact-hash attribution holds, by how the investigator obtained the suspect text. ✓: it resolves to its recorded source; ×: it does not.

Case	Concrete example	Resolves	What would fix it
Read from the store	Suspect fetched by its logged chunk id	✓	–
Retrieved copy	Passage as returned by retrieval, unmodified	✓	–
Log-recovered	Suspect copied from an answer log or screen	×	Log chunk ids, read the store
Rewritten in transit	A re-ranker or formatter alters the passage after retrieval	×	Hash before that stage
Re-derived from source	Page re-crawled, or the PDF re-extracted with another library	×	Trace the stored chunk, not the live source
Never admitted	Chunk written to the store outside the gate	×	Route every write through the gate
Shared text	Identical passage admitted from two sources	×	Evidence beyond content alone