USENIX Security '26 Artifact Appendix: Your Keywords Know Each Other: Breaking SSE with <1% Leaked Documents

Mingyu Bian[1] Jiabei Wang[1, ✉] Dandan Xu[1]
Guangyu Huang[1] Yongbin Zhou[1, ✉]

[1]*School of Cyber Science and Engineering, Nanjing University of Science and Technology, China*
{saunafish, wangjiabei, xudandan, huangguangyu, zhouyongbin}@njust.edu.cn

A Artifact Appendix

A.1 Abstract

The artifact is a Python reference implementation of WHISPER, the leakage-abuse attack on Searchable Symmetric Encryption (SSE) introduced in the paper. It bundles the WHISPER attack and its greedy and late-fusion variants, the VAL and LEAP baselines, the LLM-assisted co-occurrence pipeline, dataset parsers for six public corpora, and ten self-contained scripts that reproduce every figure and table in the paper. The artifact ships pre-built keyword and document-body caches for all six datasets, so the experiments run end-to-end without downloading any raw corpus. All experiments run on a commodity CPU machine; only the optional LLM-synthesis step requires an (inexpensive) third-party API key.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None. The artifact performs no destructive operation, contacts no live SSE service, and disables no security mechanism on the evaluator's host. It only reads the shipped public-dataset caches, runs local computation, and writes result logs under `results/`. The optional LLM-synthesis path sends *sanitized* seed text (headers, quoted replies, and contact information removed) to a third-party OpenAI-compatible API using the evaluator's own key; evaluators who prefer not to use an external service can run the structural-only variant (`multiplier=0`), which makes no network calls.

A.2.2 How to access

Permanently archived on Zenodo under the concept DOI https://doi.org/10.5281/zenodo.20355588, which always resolves to the latest published version of the artifact.

✉ Corresponding authors.

A.2.3 Hardware dependencies

No special hardware is required. A commodity x86_64 or Apple-silicon workstation with at least 16 GB of RAM and a few GB of free disk is sufficient; the attack builds dense co-occurrence matrices over the ∼30k-document corpora, which is the main memory driver. No GPU, TEE, or FPGA is needed: the `sentence-transformers` embedding model runs on CPU. A CUDA-capable GPU, if present, only speeds up the embedding step.

A.2.4 Software dependencies

Linux (e.g., Ubuntu 22.04/24.04) or macOS, and Python 3.9 or newer (developed and tested on Python 3.13). All Python packages are listed in `requirements.txt` (numpy, scipy, nltk, tqdm, sentence-transformers, torch, openai, python-dotenv, jieba). Three NLTK data packages (`stopwords`, `punkt`, `punkt_tab`) are downloaded once during set-up. Experiments that exercise the LLM-synthesis path additionally require an OpenAI-compatible HTTPS endpoint: the artifact is configured by default for **DeepSeek V4 Flash** (`deepseek-v4-flash`), a low-cost substitute for the Google Gemini 2.5 Flash model used in the paper. We verified that DeepSeek V4 Flash reproduces the paper's headline numbers within ∼1–2 percentage points, so either model is suitable.

A.2.5 Benchmarks

Six public corpora are used: Enron (corporate email), Lucene (Apache developer mailing list), Wikipedia (Simple English), LKML, QEMU, and CC-News. For each dataset the artifact ships a slim keyword/metadata cache (`pickles/<dataset>_5000.pkl`) used by the structural attacks and a gzipped document-body cache (`pickles/<dataset>_bodies.pkl.gz`) used by the LLM-synthesis path. With these caches the experiments are fully self-contained; no raw corpus download is needed.

A.3 Set-up

A.3.1 Installation

From the artifact root, in a clean Python 3.9+ environment:

1. `pip install -r requirements.txt`
2. `python -c "import nltk; nltk.download('stopwords'); nltk.download('punkt'); nltk.download('punkt_tab')"`
3. Before long experiments, pre-cache the embedding model to avoid transient HuggingFace/SSL failures: `python -c "from sentence_transformers import SentenceTransformer; SentenceTransformer('all-mpnet-base-v2')"`
4. `cp .env.example .env`, then set `DEEPSEEK_API_KEY` (the default `DEEPSEEK_BASE_URL` already points to `https://api.deepseek.com/v1`). This step is only needed for experiments that use LLM synthesis (E1–E6 below all do; the basic test can be run without a key by setting `multiplier=0`).

A.3.2 Basic Test

Run the basic functionality test:

```
python -m experiments.compare_val_whisper_leakage
```

The script loads the Enron cache (∼1 min), runs three repetitions at 0.1% leakage, and writes a timestamped log to `results/`. Expected output: a VAL query CRR of ∼17% and a WHISPER CRR of ∼40%, i.e. a strictly positive, roughly 2× gap in every run. Total time ∼7 minutes on a CPU machine.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): At 0.1% leakage, WHISPER improves the correct query recovery rate (CRR) over VAL by roughly 2–2.5× across six datasets (§6.1, Table 3 and Fig. 8). Validated by (E1).

(C2): Recovery scales with leakage, approaching near-complete recovery (exceeding or approaching 80%, up to ∼95%) by 1% leakage, and remains non-trivial even below 0.1% (§6.2, Figs. 9 and 10). Validated by (E2).

(C3): Every WHISPER component contributes to the final CRR: semantic embedding propagation dominates the average gain, LLM synthesis is complementary, and global Hungarian matching beats greedy matching (§6.3, Tables 5 and 7). Validated by (E3).

(C4): Integrating the embedding signal as an early soft prior outperforms late fusion and direct propagation (§6.3, Table 6). Validated by (E4).

(C5): Naively injecting LLM-generated documents into prior attacks (VAL, LEAP) does *not* improve them (§6.2, Table 4). Validated by (E5).

(C6): WHISPER remains effective under the padding and volume-hiding defenses evaluated in the paper, retaining a clear advantage over VAL (§6.4, Fig. 12). Validated by (E6).

A.4.2 Experiments

Each experiment below pairs the command to run (*Execution*) with the expected output and its comparison to the paper (*Results*). Every script is run from the artifact root and writes a timestamped log under `results/`. All scripts ship configured for the 0.1% leakage operating point with three repetitions (`random_seed=45`), the setting that reproduces the reference logs bundled in `results/`. Times are wall-clock on a commodity multi-core CPU with no GPU. Because the shipped LLM differs from the paper's (DeepSeek V4 Flash vs. Gemini 2.5 Flash) and each script uses a single seed, WHISPER numbers may differ from the paper by ∼1–5 pp, while VAL (deterministic) matches closely.

(E1): *Multi-dataset comparison* [5 human-min + ∼8 compute-min]. Reproduces C1 on CC-News.
Execution: `python -m experiments.multi_dataset_validation`
Results: `results/multi_dataset_<ts>.txt` reports VAL vs. Whisper query CRR at 0.1%. Expected: VAL ∼16.7%, Whisper ∼41% (paper Table 3). To cover all six datasets, set `datasets=[...]` in the script's `__main__`.

(E2): *Leakage sweep on Enron* [5 human-min + ∼7 compute-min at 0.1%]. Reproduces C2.
Execution: `python -m experiments.compare_val_whisper_leakage`
Results: `results/val_vs_whisper_leakage_<ts>.txt`. Expected at 0.1%: VAL ∼17.7%, Whisper ∼40%. For the full curve, set `leakage_rates=[0.1,...,1.0]` in `__main__` (∼1–2 h); Whisper then approaches ∼95% by 1%. `test_whisper_limits` characterizes the <0.1% regime (Fig. 10).

(E3): *Component and matching ablation on Enron* [5 human-min + ∼13 compute-min]. Reproduces C3.
Execution: `python -m experiments.ablation_study` and `python -m experiments.ablation_matching_algorithm`
Results: `results/ablation_enron_<ts>.txt` reports the five configurations of Table 5. Expected CRR, ours (paper in parentheses): VAL ∼17.7% (16.7); w/o AI ∼25.1% (24.6); w/o Co-Prop ∼26.2% (27.6); w/o LLM ∼40.0% (39.1); full ∼41.5% (41.0). The matching script reproduces Table 7: Hungarian ∼40% vs. greedy ∼30%.

(E4): *Semantic-fusion mechanism on Enron* [5 human-min + ∼7 compute-min]. Reproduces C4.
Execution: `python -m experiments.`

`compare_external_strategies`
Results: `results/external_strategies_<ts>.txt`. Expected CRR, ours (paper in parentheses): direct propagation ∼30% (30.7); late fusion ∼36% (37.3); soft prior ∼40% (40.6); the soft-prior mechanism is best (Table 6).

(E5): *Naive LLM augmentation of prior attacks* [3 human-min + ∼3 compute-min]. Reproduces C5.
Execution: `python -m experiments.compare_traditional_llm_baselines`
Results: `results/traditional_llm_<ts>.txt`. Expected: each `+LLMdocs` variant is no better than its baseline (CRR change ≤ 0), confirming that naive LLM injection does not help VAL or LEAP (Table 4).

(E6): *Defense sweep on Enron* [5 human-min + ∼22 compute-min]. Reproduces C6.
Execution: `python -m experiments.defend`
Results: `results/defense_<ts>.txt` reports VAL vs. Whisper CRR under no defense, padding (n=500), and volume hiding. Expected (VAL→Whisper): no defense ∼17.9%→41.4%; padding ∼10.4%→20.2%; volume hiding ∼13.6%→33.4%. Padding degrades both attacks the most, yet Whisper keeps a clear (∼2×) advantage over VAL under both (Fig. 12).

Two further scripts produce the implementation-section sensitivity figures: `compare_whisper_multipliers` (Fig. 6) and `compare_blend_weights` (Fig. 7), confirming the defaults $\mu = 1$ and $\alpha = 0.3$.

A.5 Notes on Reusability

Adding a new corpus only requires writing a parser in `utils/email_extraction.py` and registering its path in `MAILDIRS` in `experiments/common.py`; the rest of the pipeline picks it up automatically and caches it on first use. New attacks that follow the constructor and `attack()` interface of the reference attacks in `attacks/` drop into the experiment runners in `experiments/` without changes. The LLM-synthesis path uses a standard OpenAI-compatible client, so any compatible model can be substituted by editing `model_name` and the `DEEPSEEK_*` variables (e.g., to reproduce with the paper's original Gemini 2.5 Flash). Key knobs are exposed in each script's `__main__`: `leakage_rates`, `runs`, `random_seed`, the synthesis `multiplier`, and the embedding `external_blend_weight`; quality-aware vs. uniform known-subset sampling can be toggled via the helpers in `experiments/common.py`.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.