

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

DDR-SSE: Duplicated Retrieval of Documents for System-wide Secure Searchable Symmetric Encryption

Zichen Gui, *University of Georgia;* Simon-Philipp Merz and Kenneth G. Paterson, *ETH Zürich;* Sikhar Patranabis, *IBM Research India*

https://www.usenix.org/conference/usenixsecurity26/presentation/gui

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: DDR-SSE: Duplicated Retrieval of Documents for System-wide Secure Searchable Symmetric Encryption

Zichen Gui
University of Georgia

Simon-Philipp Merz
ETH Zürich

Kenneth G. Paterson
ETH Zürich

Sikhar Patranabis
IBM Research India

A Artifact Appendix

A.1 Abstract

DDR-SSE is a system-wide secure searchable symmetric encryption scheme. This artifact provides a roadmap for the evaluation of DDR-SSE in terms of its efficiency and resilience to leakage-abuse attacks. All results shown in Section 4 and 5 of the paper can be reproduced with the artifact.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

We use the Enron email corpus [1] to evaluate DDR-SSE as it is the only public dataset of this kind and it has been used in previous works extensively. The corpus contains emails from real people. Although it has already been sanitized, we advise consumers of the corpus not look at the individual emails directly. Our artifact do not rely on the actual content of the emails, but rather, the statistical properties of them (e.g., the number of keywords in each email).

A.2.2 How to access

The artifact can be accessed via https://doi.org/10.5281/zenodo.17965794.

A.2.3 Hardware dependencies

The artifact can be run on a normal computer. The hardware requirements are:

- CPU: Any modern CPU.
- RAM: 64 GB.
- Storage: 10 GB.

We recommend running the cryptanalysis experiments in Section 4 of our paper on a computer cluster, as these experiments can be parallelized.

[1] https://www.cs.cmu.edu/~enron/

A.2.4 Software dependencies

Our artifact requires Java (JDK 17 and above) and Python (3.8 and above). These can be downloaded and installed from https://www.oracle.com/java/technologies/downloads/ and https://www.python.org/downloads/ respectively.

A.2.5 Benchmarks

The benchmarks and cryptanalysis experiments are run using the Enron email corpus. The dataset can be downloaded from https://www.cs.cmu.edu/~enron/.

A.3 Set-up

A.3.1 Installation

The experiments require a few non-standard Python modules: nltk, numpy, scipy, sklearn, pandas, matplotlib, tqdm, packaging, pytz. These can be installed using pip (https://pypi.org/project/pip/). These modules are listed in requirements.txt, so one can simply run pip install -r requirements.txt to install all dependencies at once.

A.3.2 Basic Test

We do not provide a dedicated basic test of the artifact as it consists of several independent components. Instead, one can check if the components are functioning by running them individually using small parameters. We give more detail in Appendix A.4.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): DDR-SSE is practically efficient. This is proven by the experiment (E1 and E2) whose results are reported in Section 5 of our paper.

(C2): DDR-SSE is resilient to the state-of-the-art leakage-abuse attacks. This is demonstrated by the state-of-the-art leakage-abuse attacks (E5, E6, and E7) and our own attack (E3 and E4).

A.4.2 Experiments

The step-by-step instructions can also be found in README.md of the artifact. For the tasks where the implementation is parallelized (e.g., by using multiprocessing), we report the raw compute time without any parallelization, and the default amount of parallelization. For example, 20 compute-hour (20) means the computation takes 20 hours in total using a single process. With the default amount of parallelization, it should take about an hour.

(E1): [Download and parse the Enron emails] [5 human-minutes + 0.3 compute-hour + 6GB disk]: this step prepares the inputs for Experiments E2 and E3.
Preparation: Download the Enron email corpus. Unzip it into ./emails_raw/.
Execution: Parse the emails by running ./email_parser/email_parser.py. Use the flag --ndoc 1000 to generate small outputs for the basic test. Use flag --ndoc 400000 and --padding p for $p = 4096, 8192$, and 16384 for the full runs.
Results: The files produced are used in the subsequent steps (E2, E3, and E4).

(E2): [Benchmark DDR-SSE] [5 human-minutes + 1 compute-hour]:
Preparation: Navigate to ./DDR-SSE/. Compile the Java code.
Execution: Run Scheme.DDR_benchmark. The program takes three arguments: the number of documents ($1,000$ for the test run, $400,000$ for the full runs), the amount of padding on the encrypted documents ($4,096$ for the test run, $4096, 8192, 16384$ for the full runs), and the bucket size (400 for all runs). Use the Java runtime flag -Xmx55G to give the JVM 55GB memory.
Results: The full runs produce benchmark results in ./DDR-SSE/. Move them to ./benchmark_plots/. Run the plot script in ./benchmark_plots/ to produce Figure 4 in the paper. The other relevant performance statistics are displayed by the script.

(E3): [Leakage extraction] [5 human-minutes + 1 compute-hour + 1GB disk]:
Preparation: Navigate to ./DDR-SSE/. Compile the Java code.
Execution: Run Scheme.DDR_leakage_extraction. The program takes three arguments: the number of documents ($1,000$ for the test run, $400,000$ for the full runs), the amount of padding on the encrypted documents ($4,096$ for all runs), and the bucket size (200 for the test run, $50, 100, 200, 400$ for the full runs). Use the Java runtime flag -Xmx55G to give the JVM 55 GB memory.
Results: The program produces files describing the leakage of DDR-SSE in ./leakage/. Move the files to the folder ./leakage/exact/. These files are used in Experiment E4.

(E4): [Our leakage-abuse attack] [5 human-minutes + 18 compute-days (24) + 1GB disk]:
Preparation: Navigate to ./attack/.
Execution: Run attack_exact_batch.py. For the test run, use argument --nd 1000 --nitr 1000 --nrun 2. For the full runs, use arguments --nbkt b --pct p for $b = 50, 100, 200, 400$ and $p = 100, 95, 90, 85, 80, 75$. We recommend running these jobs on a computer cluster as there are a lot of them.
Alternatively, run script_mp.py on an ordinary computer. The default number of processes used in the script is 20.
Results: The attacks produce reports in ./attack_results/. Run the script attack_results_plot_exact.py in the folder to reproduce Figure 2 in our paper.

(E5): [The SAP attack against DDR-SSE] [5 human-minutes + 20 compute-hours (20)]:
Preparation: Navigate to ./USENIX21-sap-code-master/.
Execution:
1. Run add_experiments_to_manager.py.
2. Run manager_df.py, type w and press return.
3. Run run_pending_experiments.py (adjust the number of cores used for the jobs on line 30 of the Python script if needed).
4. Run manager_df.py. Type eat and press return to consume the attack results.

Results: Run plot5_vs_CLRZ_DDR-SSE.py to reproduce Figure 3(a) in our paper.

(E6): [The IHOP attack against DDR-SSE] [5 human-minutes + 60 compute-hours (20)]:
Preparation: Navigate to ./USENIX22-ihop-code-master/.
Execution:
1. Run add_to_manager.py.
2. Run manager.py, type w and press return.
3. Run run_from_manager.py (adjust the number of cores used for the jobs on line 131 of the Python script if needed).
4. Run manager.py. Type eat and press return to consume the attack results.

Results: Run plot5_vs_CLRZ_DDR-SSE.py to reproduce Figure 3(b) in our paper.

(E7): [The Jigsaw attack against DDR-SSE] [5 human-minutes + 2 compute-hours]:
Preparation: Navigate to ./USENIX24-jigsaw-code-master/.
Execution: Run test_against_CLRZ_DDR-SSE.py.
Results: Run generate_test_against_CLRZ_DDR-SSE.py to reproduce Figure 3(c) in our paper.

A.5 Notes on Reusability

The artifact is split into different modules to make them as reusable as possible.

- The email parser in ./email_parser/ can be adapted to

parse different datasets. It can be used to benchmark other encrypted search schemes.

- The implementation of DDR-SSE in ./DDR-SSE/ supports arbitrary databases (see the email parser for the expected input format). This makes the implementation flexible for (comparative) benchmarks in the future.
- There are two main components in DDR-SSE, namely index retrieval and document retrieval. These components are implemented separately. In the future, if there is a more efficient and/or more secure index retrieval scheme and/or document retrieval scheme, one can substitute that component in DDR-SSE easily.
- Our cryptanalysis attack in ./attack/ can be modified to attack other schemes. Only the score function (line 147 of attack_exact_batch.py) and a few auxiliary functions need to be updated.

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/.