Efficient Single-Round Obfuscation of Search and Result Patterns in Searchable Encryption

Tung Le
Virginia Tech
Blacksburg, VA, USA
tungle@vt.edu

Thang Hoang
Virginia Tech
Blacksburg, VA, USA
thanghoang@vt.edu

A Artifact Appendix

Our work FROST is a novel differentially private searchable encryption system that efficiently obfuscates both search and result patterns while providing strong resilience against all known statistical leakage-abuse attacks. FROST achieves single-round search, a key practical feature for supporting stateless system design where user requests can be routed to any available server without imposing the overhead of storing per-query state. The FROST prototype is implemented in C++ and consists of approximately 1,200 lines of code. The experiments can be conducted on a commodity server without requiring any specialized hardware. Beyond advanced performance, FROST is shown to effectively mitigate leakage of search and result patterns, as demonstrated through evaluation against state-of-the-art leakage-abuse attacks.

A.1 Abstract

The artifact includes a full implementation of FROST written in C++, along with implementations of modern leakage-abuse attacks written in Python. The execution of FROST reports detailed performance metrics for keyword search, including end-to-end latency and bandwidth overhead. For the pattern leakage attacks, the evaluation performs query recovery attacks, and measures their effectiveness in terms of accuracy, which is defined as the fraction of search queries correctly recovered out of the total number of queries. For completeness, the artifact also includes implementations of other searchable encryption counterparts using differential privacy, including CLRZ and OSSE, to enable comparative evaluation.

A.2 Description & Requirements

The source code of FROST is located in the directory `frost`, while the implementation of attacks is provided in the directory `attacks`. The file `README.md` in the main directory contains instructions for running FROST and evaluating its performance. The file `README.md` in the directory `attacks` provides guidance on executing and assessing the resilience of FROST, as well as CLRZ and OSSE, in mitigating statistical leakage-abuse attacks. In addition, the directory `scripts` contains a Python script for randomly sampling a subset of the `Wiki` dataset. All source code is released under the MIT License.

For completeness, the directory `others` contains the source code of CLRZ (retrieved from https://github.com/donnod/APOSSE) and OSSE (retrieved from https://github.com/z6shang/OSSE). The folder `FROST/others/bak` contains the source code of CLRZ with erasure coding enabled. For the evaluation, since neither FROST nor OSSE uses erasure coding, we leave this code in the repository temporarily but do not use it. The file `README.md` in this folder provides instructions for installation and execution of these schemes. The installation instructions for CLRZ and OSSE are provided in their respective folders, as explicitly noted in `FROST/others/README.md`. Note that evaluating these artifacts is optional within the scope of §A.4.

A.2.1 Security, privacy, and ethical concerns. Our implementation uses standard cryptographic libraries and other common ones that can be installed via package management tools (e.g., `apt` on Ubuntu). All datasets utilized in our experiments are publicly available and do not contain any harmful content. The experiments are performed entirely in the local environment, ensuring that no other public and private systems are affected. It neither contains any threat to the system's integrity or privacy that executes our source code nor raises any ethical concerns.

A.2.2 How to access. Our artifact is publicly available at: https://github.com/vt-asaplab/FROST/tree/CCS_2026.

A.2.3 Hardware dependencies. We used a server with 48-core Intel® Xeon®Platinum 8360Y CPU (2.40 GHz) and 512 GB RAM, running Rocky Linux 8.10. Additionally, the artifact has also been tested on Ubuntu 24.04.

A.2.4 Software dependencies. We used standard cryptographic libraries, including `OpenSSL` (https://openssl.org) for implementing pseudorandom function (PRF), `NTL` (https://libntl.org) and `GMP` (https://gmplib.org) for implementing rerandomized private information retrieval (RePIR) in our scheme. To evaluate the effectiveness of query recovery attacks, it is required to install (e.g., using the `pip3` command) several Python libraries, including `numpy`, `scipy`, `scikit-learn`, `matplotlib`, `tqdm`, `pytrends`, and `nltk`. For the Graph Matching attack, the `GraphM` package (http://projects.cbio.mines-paristech.fr/graphm) needs to be compiled to produce the executable file "`graphm`", which should then be placed in the folder `attacks/Freq_GraphM_SAP_IHOP/graphm/bin` (*not required in the scope of §A.4*). Note that our repository also includes the `GraphM` package, located in the directory `attacks/libgraphm`.

A.2.5 Benchmarks. We provided instructions for downloading and extracting the `Enron` (https://www.cs.cmu.edu/~enron) and `Wiki` (https://dumps.wikimedia.org/enwiki/latest) datasets in the file `README.md` of the main source folder. The `Lucene` dataset is only used for evaluating attacks and already enclosed with the artifact (not required to download). For `WikiExtractor`, we recommend using Python versions 3.8–3.10 to avoid errors due to compatibility issues. In addition, to run the `Jigsaw` attack on the `Wiki` dataset, it is necessary to download the data released by the authors via the Google Drive links provided at: https://github.com/JigsawAttack/JigsawAttack. Note that *to verify our main claims in §A.4, it suffices to only use the* `Enron` *dataset.*

A.3 Set Up

In this section, we outline the primary steps for setting up and building the project, followed by executing basic tests to verify its correct functionality.

A.3.1 Installation. We provided specific instructions for installing libraries and building from source code at https://github.com/vt-asaplab/FROST. The files `README.md` in the main source folder and the `attacks` folder contain detailed information. The bash script file `auto_setup.sh` can automate installation. However, it is essential to verify that no errors occur during the automated setup process. Once the installation is finished, navigate to the directory `frost` and build the source code by executing:

```
$ make -j8
```

This command produces the executable file "`main`" for running and evaluating the performance of keyword search and document update of FROST.

A.3.2 Basic test. We present a set of basic tests to confirm that the artifact for FROST evaluation functions correctly and as expected.

- For FROST, to execute a simple test with default parameters including Bloom filter size $m = 2912$ and the number of documents $N = 1024$, go to the directory `frost` and execute:
  ```
  $ ./main -d ./maildir
  ```
  Note that "`./maildir`" is the path to the dataset folder, which is obtained by downloading and extracting the `Enron` dataset. After initialization completes, enter a keyword (e.g., "security" without quotes) when prompted. It reports processing latency and bandwidth overhead of keyword search for the true positive rate TPR = 0.95 w.r.t various false positive rates FPR $\in \{0.025, 0.05, 0.1, 0.15, 0.2\}$.
- For the `Freq`, `IHOP`, `SAP`, and `GraphM` attacks, go to the folder `attacks/Freq_GraphM_SAP_IHOP`, then execute:
  ```
  $ python3 debug.py
  ```
  By default parameters, it outputs the accuracy of the `Freq` attack under the FROST defense with the `Enron` dataset, TPR = 0.95 and FPR = 0.1.
- For the `Jigsaw` attack, go to the folder `attacks/Jigsaw`, then execute:
  ```
  $ python3 test_against_countermeasure.py
  ```
  It outputs the accuracy of the `Jigsaw` attack also under the FROST defense with the `Enron` dataset, TPR = 0.95 and FPR = 0.1 by default parameters.

A.4 Evaluation Workflow

We state the major claims made in our paper and describe in detail how the key results can be validated as follows. At a high level, our experiments show that FROST achieves security guarantees equivalent to OSSE in mitigating leakage-abuse attacks, while CLRZ is significantly vulnerable. Also, FROST is more efficient than OSSE in terms of search latency and bandwidth consumption.

A.4.1 Major claims. The first two claims concern the keyword search metrics of FROST compared to OSSE, while the remaining claims are about the security guarantees of FROST and OSSE compared to CLRZ in mitigating pattern leakage attacks.

(C1): For the numbers of documents $N = 2^{10}$–2^{19} with the corresponding Bloom filter sizes m = 2912–8099, TPR = 0.95 and FPR = 0.1, the search latency of FROST is 2.9s–1683.6s, and the bandwidth overhead is 1.6 MB–144.9 MB. In comparison, the corresponding figures of OSSE are 28798.8s–806757.0s and 5.9 GB–163.9 GB, and those of CLRZ are 10.0ms–120.8ms and 0.5 KB–221.1 KB. This matches the description for Figure 11 in Section 6.1.2 of our main paper.

(C2): For Bloom filter size $m = 5201$, the number of documents $N = 2^{15}$, TPR = 0.95 and FPR increasing from 0.025 to 0.2, the search latency of FROST is 86.1s–86.7s, with the communication overhead is around 10.8 MB. In comparison, the corresponding figures of OSSE are 39466.2s–130980.0s and 8.0 GB–26.5 GB, and those of CLRZ are 11.2ms–12.5ms and 4.3 KB–26.6 KB. This matches the description for Figure 12 in Section 6.1.2 of our main paper.

(C3): For the `Freq` attack, FROST mitigates the attack accuracy to 8.7%–0.2% (*averaged over 10 iterations*) at TPR = 0.95 and FPR = 0.025–0.2, compared to 49.4% without any defense, as measured over 100K queries. OSSE provides comparable protection to FROST against the `Freq` attack, while CLRZ exhibits similar vulnerability in the absence of defenses. This matches the description for Figure 5 in Section 6.1.1 of our main paper.

(C4): For the `IHOP` attack, FROST reduces the attack accuracy to 2.1%—0.6% (*averaged over 10 iterations*) at TPR = 0.95 and FPR = 0.025–0.2, compared to 93.0% without any defense, as measured over 10K queries. OSSE provides comparable protection to FROST against the `IHOP` attack, whereas under CLRZ the attack still achieves an accuracy of 76.2%–42.6%. This matches the description for Figure 7 in Section 6.1.1 of our main paper.

(C5): For the `Jigsaw` attack, FROST decreases the attack accuracy to 3.3%–1.0% (*averaged over 10 iterations*) at TPR = 0.95 and FPR = 0.025–0.2, compared to 81.4% without any defense, as measured over 25K queries. OSSE provides a similar level of protection, reducing the attack accuracy to 7.2%–1.1%, while CLRZ remains significantly more vulnerable, with accuracy ranging from 49.3%–17.0%. This matches the description for Figure 9 in Section 6.1.1 of our main paper.

We do not include Figures 5b, 6, 7b, 8, 9b, 10, 13, 14, and 15 in our major claims because they exhibit trends similar to those already covered by the major claims, but are evaluated on larger datasets. Nevertheless, all of these results can be reproduced using the parameters provided in the corresponding figure descriptions in the main paper.

A.4.2 Experiments. Each of the following experiments is to verify the corresponding claim in §A.4.1. We estimate only computer time and omit person-time, as the required manual effort mainly involves entering commands and reading results. Note that attack-related experiments can be parallelized by running multiple processes concurrently to fully utilize multi-core CPUs without impacting the results.

(E1): *[Estimated time is about 3 days, assuming that each process is executed sequentially]:* Go to the directory `frost`, then

execute the following with the argument value followed by `-m` in $\{2912, 3745, 4676, 5810, 7273, 8099\}$ to configure Bloom filter size m, and the argument value followed by `-n` in $\{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{19}\}$ to configure the corresponding number of documents N:

```
$ ./main -d ./maildir -m 2912 -n 1024
```

The above command runs FROST with parameters Bloom filter size $m = 2912$, and the number of documents $N = 1024 = 2^{10}$. Next, wait for the initialization process to finish, then enter a keyword to perform a search. It outputs the end-to-end latency of keyword search and communication cost with varied FPRs in the range 0.025–0.2.

Similarly, for OSSE, go to the directory `others/OSSE/code` and execute the following command with the argument value followed by `-n` in $\{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{19}\}$:

```
$ python3 dp_sse_bench.py -n 1024 -t 0.95 -f 0.1
```

And for CLRZ:

```
$ java -Xmx64g org.crypto.sse.TestLocalRR2Lev -d
./maildir -n 1024 -t 0.95 -f 0.1
```

(E2): *[Estimated time is approximately 2 hours]:* Go to the directory `frost`, then execute the following:

```
$ ./main -d ./maildir -m 5201 -n 32768
```

The above command runs FROST with parameters Bloom filter size $m = 5201$, and the number of documents $N = 32768 = 2^{15}$. Next, wait for the initialization process to finish, then enter a keyword to perform a search. It outputs the delay of keyword search and bandwidth cost with varied FPRs in the range 0.025–0.2.

Similarly, for OSSE, go to the directory `others/OSSE/code` and execute the following command with the argument value followed by `-f` in $\{0.025, 0.05, 0.1, 0.15, 0.2\}$:

```
$ python3 dp_sse_bench.py -n 32768 -t 0.95 -f
0.025
```

And for CLRZ:

```
$ java -Xmx64g org.crypto.sse.TestLocalRR2Lev -d
./maildir -n 32768 -t 0.95 -f 0.025
```

(E3): *[Estimated time is about 3 days, assuming that all processes are run in parallel]:* Go to the directory `attacks/Freq_GraphM_SAP_IHOP`, then execute the following with the argument accompanied by `--fpr` in $\{0.025, 0.05, 0.1, 0.15, 0.2\}$:

```
$ python3 debug.py --nqr 100000 --nkw 1000 --tpr
0.95 --fpr 0.025 --dataset enron-full --att freq
--defense frost
```

The above command runs the `Freq` attack on the `Enron` dataset with 1000 keywords for 100K queries, using FROST as the defense mechanism with TPR = 0.95 and FPR = 0.025. It reports the accuracy of the `Freq` attack across 10 iterations, along with the average accuracy. The argument followed by `--nkw` is equivalent to the value of $|W|$ in our main paper (i.e., the keyword universe set size).

Note that in this setting the argument followed by `--dataset` (dataset name) can be "`enron-full`", "`lucene`", and "`wiki_sec`" (see available datasets in the folder `Freq_GraphM_SAP_IHOP/datasets_pro`). The argument followed by `--att` (attack technique) can be "`freq`", "`ihop`", "`sap`", and "`graphm`". We can also change the argument followed by `--defense` (defense mechanism) to "`clrz`", "`osse`", "`frost`", and "`none`" (no defense) to evaluate their effectiveness against different statistical leakage-abuse attacks. When the defense is configured as "`none`", it is used to evaluate attack accuracy under the "No defense" setting in Figures 5–10.

(E4): *[Estimated time is about 7 days, assuming that all processes are run in parallel]:* Go to the directory `attacks/Freq_GraphM_SAP_IHOP`, then execute the following with the argument accompanied by `--fpr` in $\{0.025, 0.05, 0.1, 0.15, 0.2\}$:

```
$ python3 debug.py --nqr 10000 --nkw 1000 --tpr
0.95 --fpr 0.025 --dataset enron-full --att ihop
--defense frost
```

The above command runs the `IHOP` attack on the `Enron` dataset with 1000 keywords for 10K queries, applying FROST as the defense mechanism with TPR = 0.95 and FPR = 0.025. It reports the accuracy of the `IHOP` attack across 10 iterations, along with the average accuracy.

(E5): *[Estimated time is about 5 days, assuming that all processes are run in parallel]:* Go to the directory `attacks/Jigsaw`, then execute the following with the argument accompanied by `--fpr` in $\{0.025, 0.05, 0.1, 0.15, 0.2\}$:

```
$ python3 test_against_countermeasure.py --nqr
25000 --nkw 1000 --tpr 0.95 --fpr 0.025 --dataset
enron --defense frost
```

The above command runs the `Jigsaw` attack on the `Enron` dataset with 1000 keywords for 25K queries, employing FROST as the defense mechanism with TPR = 0.95 and FPR = 0.025. It reports the accuracy of the `Jigsaw` attack across 10 iterations. Note that in this setting the argument followed by `--dataset` (dataset name) can be "`enron`", "`lucene`", and "`wiki`". We can also change the argument followed by `--defense` (defense mechanism) to "`clrz`", "`osse`", "`frost`", and "`none`" (no defense) to evaluate their effectiveness against the `Jigsaw` attack.

A.5 Notes on Reusability

We implemented FROST protocols, including setup, keyword search, and document update as described in the body of the paper. Our FROST supports oblivious search with differential privacy, offering configurable true positive and false positive rates. This is an academic proof-of-concept prototype which has not undergone thorough code review and is not intended for production use.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926.