d-DSE: Distinct Dynamic Searchable Encryption Resisting Volume Leakage in Encrypted Databases

Dongli Liu and Wei Wang, *Huazhong University of Science and Technology;* Peng Xu, *Huazhong University of Science and Technology, Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, JinYinHu Laboratory, and State Key Laboratory of Cryptology;* Laurence T. Yang, *Huazhong University of Science and Technology and St. Francis Xavier University;* Bo Luo, *The University of Kansas;* Kaitai Liang, *Delft University of Technology*

https://www.usenix.org/conference/usenixsecurity24/presentation/liu-dongli

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: d-DSE: Distinct Dynamic Searchable Encryption Resisting Volume Leakage in Encrypted Databases

Dongli Liu[1], Wei Wang[1,✉], Peng Xu[2,3,4,✉], Laurence T. Yang[1,5], Bo Luo[6], and Kaitai Liang[7]

[1]*Huazhong University of Science and Technology*
[2]*Hubei Key Laboratory of Distributed System Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology*
[3]*JinYinHu Laboratory*
[4]*State Key Laboratory of Cryptology*
[5]*St. Francis Xavier University*
[6]*The University of Kansas*
[7]*Delft University of Technology*

✉*Corresponding authors: {viviawangwei, xupeng}@hust.edu.cn*

A Artifact Appendix

A.1 Abstract

Dynamic Searchable Encryption (DSE) has emerged as a solution to efficiently handle and protect large-scale data storage in encrypted databases (EDBs). Volume leakage poses a significant threat, as it enables adversaries to reconstruct search queries and potentially compromise the security and privacy of data. Padding strategies are common countermeasures for the leakage, but they significantly increase storage and communication costs. In this work, we develop a new perspective on handling volume leakage. We start with *distinct search* and further explore a new concept called *distinct* DSE (*d*-DSE).

For applying the 'Artifacts Available' badge of *d*-DSE, we release our test code on GitHub. The artifact appendix includes all necessary information to access and establish our artifacts via GitHub resource. Since our proposal sharply reduces the communication cost as compared to padding strategies on the top of DSE, we would like to apply the 'Functional' badge for our main test: the comparison of the total search time and communication costs of BF-SRE, AURA^P, and MITRA^P without deletion. During the functional test, we can see that the communication cost of BF-SRE has 14.92x and 30.41x advantage over AURA^P and MITRA^P on the Crime dataset, which is aligned with our conclusion about the communication advantages overall schemes.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our experiment does not have risks for evaluators while executing artifact to their machines security, data privacy or others ethical concerns. The test data is public available, and the aspects of test are not related to the system vulnerability.

A.2.2 How to access

Our artifact stable URL is:

https://github.com/jd89j12dsa/ddse/tree/AEversion

Codes are available in GitHub via the command:

```
$git clone https://github.com/jd89j12dsa/ddse.git
$git checkout AEversion
```

The command can get our code in the Ubuntu system, and the underling README.md can guide evaluators establish the experiment.

Quick Setup. We would like to thank reviewers for quick setup via Docker (version 18.09.7) on Ubuntu Server 16.04 x64. In the 'ddse' folder, the following command can build the docker container to test our codes:

```
$cd ./ddse/dockerfile
$sudo docker build -t test ./
$sudo docker run -name testddse -p 80:80 -it test
(in docker container)
root@2882cbe0c7a4:/ddse$ nohup mongod &
```

After establishment, we can follow instructions from Section A.3.2 and A.4 to test experiments. For manual installation, we can follow instructions from Section A.3 and check hardware/software dependencies from Section A.2.3 and A.2.4.

A.2.3 Hardware dependencies

The artifact does not require specific hardware features. Most computers can download code from the GitHub and run it at local. We strongly recommend 128GB memory due to the large space requirements of padding strategies compared to d-DSE. The hardware dependencies can be established based on what we used: Intel® Xeon Gold 5120 CPU @ 2.20GHz, 128GB, and Dell RERC H730 Adp SCSI Disk Device.

A.2.4 Software dependencies

We require the following software on Ubuntu Server 16.04 x64 before all experiments.

- Python v3.6.3 (corresponded with pip3.6)
- MySQL v14.14 Distribute 5.7.33
- MongoDB v2.6.10
- g++-7 (7.5.0)
- cmake 3.17
- openssl 1.0.2
- Apache Thrift 0.13.0

The Python v3.6.3, pip3.6, openssldev, cmake 3.17, and Apache Thrift 0.13.0 should be installed manually, using the following instruction:

#install Python v3.6.3
```
  $wget https://www.python.org/ftp/python/
3.6.3/Python-3.6.3.tar.xz
  $tar -xvf Python-3.6.3.tar.xz
  $cd Python-3.6.3
  $./configure
  $make -j
  $make install
  $ln -s /usr/local/bin/python3.6
/usr/local/bin/python
```
#install pip3.6
```
  $wget https://bootstrap.pypa.io/pip/3.6/get-pip.py
  $python3.6 get-pip.py
  $ln -s /usr/local/bin/pip3.6 /usr/local/bin/pip
```
#install the openssldev libraries:
```
  $sudo apt-get install libssl-dev
```
#install Cmake 3.17
```
  $wget https://github.com/Kitware/CMake/
releases/download/v3.17.0-rc3/cmake-3.17.0-rc3.tar.gz
  $tar -xvf cmake-3.17.0-rc3.tar.gz
  $cd cmake-3.17.0-rc3
  $./bootstrap
  $make -j
  $make install
```
#install Apache Thrift 0.13.0
```
  $wget https://archive.apache.org/
dist/thrift/0.13.0/thrift-0.13.0.tar.gz
  $tar -xvzf thrift-0.13.0.tar.gz
  $cd thrift-0.13.0
  $./configure
  $make -j
  $make install
```

A.2.5 Benchmarks

In the 'DB_gen' folder of our GitHub codes, we provide the Crimes, Wikipedia, and Enron datasets dumped from MongoDB during our experiments.

The following code cand restore dumped dataset in MongoDB:
```
  $cd ./DB_Gen
  $mongorestore -db Crime_USENIX_REV
./Crime_USENIX_REV
  $mongorestore -db Enron_USENIX ./Enron_USENIX
  $mongorestore -db Wiki_USENIX ./Wiki_USENIX
  $cd ../
```

A.3 Set-up

A.3.1 Installation

Followed the GitHub code, we summarize the setup for BF-SRE, MITRA^P, AURA^P, SEAL, and ShieldDB in our experiment.

BF-SRE: After locating the 'BF-SRE' folder, install the python-C models for BF-SRE with pathos package:
```
  $sudo python3.6 setup_DDSE2.py install
  $sudo python3.6 setup_Diana.py install
  $cd ./SRE
  $sudo python3.6 cSRE_setup.py install
  $cd ..
  $pip3.6 install -trusted-host pypi.org
-trusted-host pypi.python.org -trusted-host
files.pythonhosted.org pathos
```
MITRA^P: After locating the 'Compare_MITRAp' folder, install the python-C models:
```
  $sudo python3.6 setup_MITRAPP.py install
  $sudo python3.6 setup_DianaM.py install
```
AURA^P: After locating the 'Compare_AURAp' folder, build the C program from Aura:
```
  $cd ./build
  $cmake ..
  $make
```
SEAL: After locating the 'Compare_Seal' folder, the third party (Boost) from CCS18 should be installed first for the path-ORAM, specifically:
```
  $wget https://archives.boost.io/release/
1.64.0/source/boost_1_64_0.tar.gz
  $tar -xzvf boost_1_64_0.tar.gz
  $cd boost_1_64_0/
  $./bootstrap.sh -prefix=/usr/local
```

```
$./b2
$sudo ./b2 install
$cd ..
```

Next, we build the python-C models:

```
$sudo python3.6 setup_OMAP.py install
$sudo python3.6 setup_ORAM.py install
```

ShieldDB: After locating the 'Compare_ShieldDB' folder, we should first install the ShieldDB with its related dependency (RocksDB v6.22.1 and gflags v2.2.2):

```
  $git clone https://github.com/MonashCyber
securityLab/ShieldDB.git
  $apt-get install build-essential     libsnappy-dev
zlib1g-dev libbz2-dev libgflags-dev liblz4-dev
  $git clone https://github.com/facebook/rocksdb.git
  $cd rocksdb
  $git checkout v6.22.1
  $mkdir build && cd build
  $cmake ..
  $make
  $cd ..
  $export CPLUS_INCLUDE_PATH= ${CPLUS_INCLUDE_PATH}
${CPLUS_INCLUDE_PATH:+:}`pwd`include
  $export LD_LIBRARY_PATH= ${LD_LIBRARY_PATH}
${LD_LIBRARY_PATH:+:}`pwd`/build/
  $export LIBRARY_PATH=${LIBRARY_PATH}
${LIBRARY_PATH:+:}`pwd`/build/
  $apt-get install python-virtualenv python-dev
  $virtualenv pyrocks_test
  $cd pyrocks_test
  $../bin/active
  $ pip3.6 install -trusted-host pypi.org
-trusted-host pypi.python.org -trusted-host
files.pythonhosted.org Cython==0.29.35
  $pip3.6 install -trusted-host pypi.org
-trusted-host pypi.python.org -trusted-host
files.pythonhosted.org python-rocksdb==0.7.0
  $ cd ..
```

After install ShieldDB, we should move the setting program into the 'ShiledDB' folder:

```
$mv ./ShieldDB_Supporter.py ./ShieldDB/
$mv ./Gen.sh ./ShieldDB/
$cd ./ShieldDB/
$mkdir ./Var/
```

A.3.2 Basic Test

In our d-DSE program, we provide a tiny dataset 'Crime_USENIX_REV_Toy' to check our test codes run correctly. The dataset should reload into MongoDB at first:

```
  $cd ./DB_Gen
  $mongorestore -db Crime_USENIX_REV_TOY
./Crime_USENIX_REV_TOY
  $cd ../
```

In the 'BF-SRE', 'Compare_MITRAp', and 'Compare_Seal' folder, we provide the shell code 'check.sh' to check the correctness of BF-SRE, MITRA^P, AURA^P, and SEAL, respectively. For ShieldDB, we use their GitHub example program to check potential bugs.

To run the shell code, we use:

```
$ sh check.sh
```

For AURA^P, the shell code is located at 'Compare_AURAp /build' folder. We first use the python code in 'dataset' folder to translate the document data into the input of AURA^P program:

```
$cd dataset
$python3.6 Aura_plan.py Crime_USENIX_REV_TOY
$cd ..
$sh ./check.sh
```

At the end of the check programs, they will create a 'result' folder in the same folder as the shell code.

The result file is a text document including each time and communication cost related to the keyword. The main fields of the document include:

`Keyword`: the keyword name (e.g., F180) that we count their ;

`Total_search`: the time cost for the keyword search on encrypted data;

`result_len`: the number of returned entries (e.g, 256 for the padding of $x = 4$ factor);

`comm_size`: the bytes of the returned search result.

A.4 Evaluation workflow

A.4.1 Major Claims

Since our main contribution is the lower communication cost compared with padding strategies on the top of DSE, we claim the main result in our evaluation:

(C1) : d-DSE achieves the less time cost than the MITRA^P and AURA^P while saving at least 6.36x communication costs. This is proven by the experiment (E1) described in Section 6.3.1 para1 whose results are illustrated in Fig 4.

A.4.2 Experiments

(E1): [360 human-minutes + 1 compute-hour + 5GB disk + 32GB RAM]: The experiment tests the search time and communication cost of BF-SRE, MITRA^P, and AURA^P on the Crime dataset.

How to: We can process the 'test.sh' shell code in the folders for BF-SRE, MITRA^P, and AURA^P to compare their '`comm_len`' result. As expected, the communication cost of BF-SRE in searching for keywords with the largest Keyword Volume will be at least 6.36 times lower than that of MITRA^P and AURA^P.

Preparation: The process environment is same as the Basic Test.

Execution: The execution code for BF-SRE and MITRA^P, is:

```
$ sh test.sh
```

where the shell code is in 'BF-SRE' and 'Compare_MITRAp' folder, respectively.

For AURA^P, the shell code is located at 'Compare_AURAp/build' folder. We use the 'Aura_plan.py' in 'dataset' folder to generate the input of AURA^P program:

```
$cd dataset
$python3.6 Aura_plan.py Crime_USENIX_REV
$cd ..
$sh ./test.sh
```

Results: As a result, BF-SRE will writes each keyword search result in the text file located at:
`BF-SRE/Result/Search/BF-SRE/0/Crime_USENIX_REV /d0/24021101`.
MITRA^P writes at:
`Compare_MITRAp/Result/Search/MITRAPP/Crime_ USENIX_REV/d0/240206`.
AURA^P writes at:
`Compare_AURAp/build/result`
The keyword search results are output in descending order according to their Keyword Volume. Base on the bytes returned on keyword 'F730', the length of BF-SRE are nearly 85.75KB with 14.92x and 30.41x advantage over AURA^P and MITRA^P, which aligns our summary on the communication reduce.

A.5 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/usenixsec2024/.