USENIX Security '25 Artifact Appendix: Engorgio: An Arbitrary-Precision Unbounded-Size Hybrid Encrypted Database via Quantized Fully Homomorphic Encryption

Song Bian[1], Haowen Pan[1], Jiaqi Hu[1], Zhou Zhang[1], Yunhao Fu[1], Jiafeng Hua[2], Yi Chen[3], Bo Zhang[3], Yier Jin[4], Jin Dong[3], and Zhenyu Guan[1]*

[1]Beihang University, [2]Huawei Technology
[3]Beijing Academy of Blockchain and Edge Computing
[4]University of Science and Technology of China

A Artifact Appendix

A.1 Abstract

Engorgio is a hybrid encrypted database based on quantized fully homomorphic encryption. Engorgio can efficiently evaluate fast and arbitrary-precision homomorphic filtering, sorting and complex aggregation algorithms that enable a variety of SQL queries to be applied over FHE ciphertexts. This artifact allows users to evaluate the availability and runtime latency of Engorgio across various benchmarks.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The evaluation of our artifact does not introduce any risks to machine security, data privacy, or ethical concerns.

A.2.2 How to access

Our implementation is available on Zenodo with the link https://zenodo.org/records/14730651.

A.2.3 Hardware dependencies

The experimental results detailed in the manuscript were derived from evaluations performed using an Intel Xeon Gold 6226R processor with 512 GB of RAM in a single-thread environment.

A.2.4 Software dependencies

To evaluate the artifact, the specific OS and software packages required include Ubuntu 20.04 as the operating system, along with essential development tools like build-essential, g++-10, apt-utils, ca-certificates, as well as version control (git), build tools (cmake>=3.16), and cryptographic library (libgmp-dev, libntl-dev).

A.2.5 Benchmarks

SIFT dataset.

A.3 Set-up

A.3.1 Installation

Engorgio can be built (out-of-source) by executing following commands:

```
$ cd Engorgio
$ mkdir build
$ cd build
$ cmake ..
$ make
```

A.3.2 Basic Test

Following the instructions above, nine output binaries will be generated in the `build/bin/` directory. You can run these binaries by:

```
$./bin/comparison_test
$./bin/sort_test
$./bin/sync_test
$./bin/topk_test
$./bin/relational_query_test
$./bin/vectorized_query_test_1
$./bin/vectorized_query_test_2
$./bin/hybrid_query_test_1
$./bin/hybrid_query_test_2
```

*Corresponding author.

A.4 Evaluation workflow

A.4.1 Major Claims

We enumerate the major claims of Engorgio as follows:

(C1): Engorgio achieves 28×–854× faster homomorphic comparison, 65×–687× faster homomorphic sorting, 42×–4,595× faster homomorphic synchronization, and 4×–58× faster homomorphic Top-k over the state-of-the-art (SOTA) solutions. This is proven by experiments (E1, E2, E3 and E4) whose results are illustrated in Figure 5.

(C2): Engorgio is capable of evaluating efficient relational, vectorized, and hybrid query benchmarks. This is proven by experiments (E5, E6 and E7), whose results are illustrated in Figure 7.

A.4.2 Experiments

(E1): [Homomorphic Comparison] [3 human-minutes + 30 compute-minutes]: In this experiment, we perform the homomorphic comparison operator in Engorgio, which is the basic operator for a hybrid encrypted database.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing `./bin/comparision_test`.
Results: The runtime latency for the 8-bit precision homomorphic comparison operators (both relational operator and equality operator) is less than 1 ms, while the latency for the 16-bit precision is around 1.5 ms, for the 32-bit precision, it is approximately 3 ms, and for the highest precision of 64 bits, the latency reaches approximately 8 ms. These evaluations were performed using one thread and produced accurate results as shown in Figure 5(a) and Figure 5(b).

(E2): [Homomorphic Sorting] [3 human-minutes + 25 compute-hours]: In this experiment, we perform the homomorphic sorting operator in Engorgio, which is the basic operator for data ordering and vector search.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing `./bin/sort_test`.
Results: The runtime latency for the 8-input homomorphic sorting is around 4.8×10^2 ms, while the latency for the 64-input homomorphic sorting is around 1.44×10^4 ms, for the 1024-input homomorphic sorting is around 5.74×10^5 ms, and for the 32768-input homomorphic sorting, the latency reaches approximately 3.79×10^7 ms. These evaluations were performed using one thread and produced accurate results as shown in Figure 5(c).

(E3): [Homomorphic Synchronization] [3 human-minutes + 3 compute-hours]: In this experiment, we perform the homomorphic synchronization operator in Engorgio.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing `./bin/sync_test`.
Results: The runtime latency for the 8-input homomorphic synchronization is around 2 ms, while the latency for the 64-input homomorphic sorting is around 95 ms, for the 1024-input homomorphic sorting is around 1×10^4 ms, and for the 8192-input homomorphic sorting, the latency reaches approximately 1.42×10^5 ms. These evaluations were performed using one thread and produced accurate results as shown in Figure 5(d).

(E4): [Homomorphic Topk] [3 human-minutes + 12 compute-hours]: In this experiment, we perform the homomorphic synchronization operator in Engorgio.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing `./bin/topk_test`.
Results: The runtime latency for the Top-1 in 256-input is around 1.6×10^4 ms, while the Top-4 in 256-input is around 4.2×10^4 ms, for the Top-8 in 256-input is around 5.2×10^4 ms, and for the Top-16 in 256-input, the latency reaches approximately 6×10^4 ms. These evaluations were performed using one thread and produced accurate results as shown in Figure 5(e).

(E5): [Relational Query] [3 human-minutes + 1.5 compute-hours]: In this experiment, we perform the end-to-end relational SQL query in Engorgio.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing `./bin/relational_query_test`.
Results: The runtime latency for executing TPC-H Q1 and TPC-H Q12 on an encrypted database with varying row sizes ($2^{11}, 2^{12}, 2^{13}$, and 2^{14}) is as follows: TPC-H Q1 takes approximately 32 seconds, 66 seconds, 134 seconds, and 270 seconds, respectively. TPC-H Q12 takes approximately 28.5 seconds, 56.8 seconds, 114 seconds, and 229 seconds, respectively. These evaluations were performed using one thread and produced accurate results as shown in Figure 7(a) and Figure 7(b).

(E6): [Vectorized Query] [3 human-minutes + 30 compute-hours]: In this experiment, we perform the end-to-end vectorized query in Engorgio.
Preparation: Following the instructions in Appendix A.3.1.
Execution: Running the binary file in the `build/bin/` directory by executing:
`./bin/vectorized_query_test_1`
`./bin/vectorized_query_test_2`.
Results: The runtime latency for executing vectorized query VQ1 and VQ2 on an encrypted database with

varying row sizes ($2^7, 2^8, 2^9$, and 2^{10}) is as follows: VQ1 takes approximately 20 seconds, 48 seconds, 113 seconds, and 266 seconds, respectively. VQ2 takes approximately 23 seconds, 64 seconds, 150 seconds, and 340 seconds, respectively. These evaluations were performed using one thread and produced accurate results as shown in Figure 7(c) and Figure 7(d).

(E7): [Hybird Query] [3 human-minutes + 30 compute-hours]: In this experiment, we perform the hybrid query experiment in Engorgio.

Preparation: Following the instructions in Appendix A.3.1.

Execution: Running the binary file in the `build/bin/` directory by executing:

```
./bin/hybrid_query_test_1
./bin/hybrid_query_test_2.
```

Results: The runtime latency for executing hybrid query HQ1 and HQ2 on an encrypted database with varying row sizes (2^6, 2^8, 2^{10}, and 2^{12}) is as follows: HQ1 takes approximately 15.3 seconds, 105 seconds, 600 seconds, and 3313 seconds, respectively. HQ2 takes approximately 9.3 seconds, 50 seconds, 259 seconds, and 1270 seconds, respectively. These evaluations were performed using one thread and produced accurate results as shown in Figure 7(e).

A.5 Notes on Reusability

None

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/usenixsec2025/.