USENIX Security '26 Artifact Appendix: InstantOMR: Oblivious Message Retrieval with Low Latency and Optimal Parallelizability

Haofei Liang
Shanghai Jiao Tong University

Zeyu Liu
Yale University

Eran Tromer
Boston University

Xiang Xie
Primus Labs

Yu Yu
Shanghai Jiao Tong University

A Artifact Appendix

A.1 Abstract

This artifact contains our end-to-end code for our Oblivious Message Retrieval (OMR) construction, InstantOMR. InstantOMR is a construction based on the TFHE homomorphic encryption system. Our artifact demonstrates the performance of InstantOMR when implemented with the Primus-fhe library. Below, we discuss how to use our code and our main claims about its efficiency.

A.2 Description & Requirements

This section lists all the information necessary to recreate the same experimental setup we have used to run your artifact.

A.2.1 Security, privacy, and ethical concerns

We do not see any direct security, privacy, or ethical concerns. Specifically, our code can be run independently and does not need to interact with any other software or environment outside.

A.2.2 How to access

The artifact is available at Zenodo: https://doi.org/10.5281/zenodo.17959835.

A.2.3 Hardware dependencies

Our code can be compiled and run without specific hardware. However, one thing to note is that we benchmark InstantOMR scheme using Google Compute Cloud "c3d-highcpu-360", equipped with 180 cores and AVX-512 instructions. The numbers reported in our main claims are obtained using this instance. Other instances can also be used, but may result in a different performance.

A.2.4 Software dependencies

We have tested our code on Windows (MSVC toolchain) and Ubuntu (20.04/22.04).

To compile and run our code, the first step is to install build tools. On Windows, please install Visual Studio C++ Build tools. On Ubuntu and Debian, please install build-essential according to the instructions below:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
```

macOS has **not** been tested or adapted. It may not run or reproduce results correctly on macOS. If you try on macOS, please run **without** `+nightly` and **without** `-features="nightly"` first; the build may still fail.

A.2.5 Benchmarks

None.

A.3 Set-up

This project builds on stable Rust by default. The nightly toolchain is **optional** and only required if you want AVX-512 acceleration via the `nightly` feature. Here are the steps to install Rust after installing the build tools in section A.2.4:

1. Install Rust using rustup (the recommended Rust installer):

   ```
   $ curl --proto '=https' --tlsv1.2 -sSf https://
       sh.rustup.rs | sh
   ```

   On Windows, one can download and run the installer "rustup-init.exe" from https://rust-lang.org/tools/install/.

2. After installation, verify Rust is installed correctly:

   ```
   $ rustc --version
   $ cargo --version
   ```

3. (Optional) Install the nightly toolchain (only needed for AVX-512 support):

```
$ rustup toolchain install nightly
```

4. (Optional) Verify the nightly toolchain is available:

```
$ rustc +nightly --version
```

For more information, see the Rust installation guide.

A.3.1 Installation

1. Visit https://doi.org/10.5281/zenodo.17959835 and download the artifacts `tfhe-omr.zip`.

2. Unarchive the `tfhe-omr.zip` file on your local computer.

3. Run the following commands `cd tfhe-omr`.

A.3.2 Basic Test

Remark. If `+nightly` fails in your environment, please use the stable commands (without `+nightly` and without `-features="nightly"`). `+nightly` is optional for AVX-512 acceleration and may not build if toolchain/CPU feature flags do not match; then, stable commands are sufficient and recommended.

1. Build the InstantOMR example.

```
$ cargo +nightly build --package omr_core --
    example omr --features="nightly" --release
```

2. Run InstantOMR example with single thread and single payload.

```
$ cargo +nightly run --package omr_core --
    example omr --features="nightly" --release
    -- --thread-count 1 --payload-count 1
```

3. Run InstantOMR example with 4 threads and 16 payloads.

```
$ cargo +nightly run --package omr_core --
    example omr --features="nightly" --release
    -- --thread-count 4 --payload-count 16
```

It should be noted that the maximum payload-count supported by InstantOMR example is 65536, and the maximum thread-count is recommended to be the number of cores of the machine. For example, a CPU with 8 cores and 16 threads can run 16 threads at the same time, but the optimal operating efficiency is achieved at 8 threads.

A.4 Evaluation workflow

A.4.1 Major Claims

Our major claims about our code are its server (detector) runtime, including its latency, throughput, and parallelizability, using our environment Google Compute Cloud "c3d-highcpu-360", equipped with 180 cores, 708GB of memory, and AVX-512 instructions. In particular:

(C1): Latency: with a single message, InstantOMR server finishes in $\sim$274 milliseconds.

(C2): Throughput: with 2^{16} messages, InstantOMR server finishes in $\sim$5 hours

(C3): Parallelizability: with c messages, for c being the number of cores in the server's machine, InstantOMR server finishes in $\sim$274 ms (nearly identical to the runtime of a single message).

A.4.2 Experiments

(E1): *[Latency] [10 human-minutes + 5 compute-seconds]: Run* InstantOMR *example with single thread and single payload.*

Preparation: *Build the* InstantOMR *example.*

```
$ cargo +nightly build --package omr_core --
    example omr --features="nightly" --release
```

Execution: *Run* InstantOMR *example with single thread and single payload.*

```
$ cargo +nightly run --package omr_core --
    example omr --features="nightly" --release
    -- --thread-count 1 --payload-count 1
```

Results: *The detect time* $\sim$ 243*ms of the example result shows* InstantOMR *has a low latency described in the paper.*

```
num threads: 1
all payloads count: 1
2025-04-14T14:08:22.064673Z DEBUG ThreadId(01)
    omr: Generating secret key pack...
2025-04-14T14:08:22.065688Z DEBUG ThreadId(01)
    omr: Generating sender and detector...
2025-04-14T14:08:22.719183Z DEBUG ThreadId(01)
    omr: Generating clues...
2025-04-14T14:08:22.720238Z INFO ThreadId(01)
    omr: gen clues time: 945.4us
2025-04-14T14:08:22.720301Z DEBUG ThreadId(01)
    omr: Generating payloads...
2025-04-14T14:08:22.720346Z INFO ThreadId(01)
    omr: gen payloads time: 7.9us
2025-04-14T14:08:22.720628Z DEBUG ThreadId(01)
    omr: Detecting...
[elapsed: 00:00:00] [########################]
    1/1 [eta: 00:00:00] [0s]
2025-04-14T14:08:22.964331Z DEBUG ThreadId(01)
    omr: Detect done
```

```
2025-04-14T14:08:22.964418Z INFO ThreadId(01)
    omr: detect time: 243.6431ms
2025-04-14T14:08:22.964466Z INFO ThreadId(01)
    omr: detect time per message: 243.6431ms
2025-04-14T14:08:22.964819Z INFO ThreadId(01)
    omr: encode indices times: 146.3us
2025-04-14T14:08:22.964878Z INFO ThreadId(01)
    omr: encode indices times per ciphertext:
    48.766us
2025-04-14T14:08:22.965031Z INFO ThreadId(01)
    omr: encode pertinent payloads time: 79.5us
2025-04-14T14:08:23.011517Z INFO ThreadId(01)
    omr: decode time: 46.4251ms
2025-04-14T14:08:23.011637Z INFO ThreadId(01)
    omr: All done
```

(E2): *[Throughput] [10 human-minutes + 5 compute-hours]: Run* InstantOMR *example with single thread and 65536 payloads*

Preparation: *Build the* InstantOMR *example.*

```
$ cargo +nightly build --package omr_core --
    example omr --features="nightly" --release
```

Execution: *Run* InstantOMR *example with single thread and 65536 payloads.*

```
$ cargo +nightly run --package omr_core --
    example omr --features="nightly" --release
    -- --thread-count 1 --payload-count 65536
```

Results: *The detect time* $\sim$ 15340*s of the example result shows the throughput of* InstantOMR*.*

```
num threads: 1
all payloads count: 65536
2025-04-14T14:10:57.838257Z DEBUG ThreadId(01)
    omr: Generating secret key pack...
2025-04-14T14:10:57.838995Z DEBUG ThreadId(01)
    omr: Generating sender and detector...
2025-04-14T14:10:58.444275Z DEBUG ThreadId(01)
    omr: Generating clues...
2025-04-14T14:11:57.382862Z INFO ThreadId(01)
    omr: gen clues time: 58.9381541s
2025-04-14T14:11:57.383002Z DEBUG ThreadId(01)
    omr: Generating payloads...
2025-04-14T14:11:57.403619Z INFO ThreadId(01)
    omr: gen payloads time: 20.5511ms
2025-04-14T14:11:57.403832Z DEBUG ThreadId(01)
    omr: Detecting...
[elapsed: 04:15:40] [########################]
    65,536/65,536 [eta: 00:00:00] [0s]
2025-04-14T18:27:37.612223Z DEBUG ThreadId(01)
    omr: Detect done
2025-04-14T18:27:37.612318Z INFO ThreadId(01)
    omr: detect time: 15340.2083335s
2025-04-14T18:27:37.612369Z INFO ThreadId(01)
    omr: detect time per message: 234.073003ms
2025-04-14T18:27:41.094382Z INFO ThreadId(01)
    omr: encode indices times: 3.4819539s
2025-04-14T18:27:41.094491Z INFO ThreadId(01)
    omr: encode indices times per ciphertext:
    696.39078ms
2025-04-14T18:28:05.354507Z INFO ThreadId(01)
    omr: encode pertinent payloads time:
    24.2598764s
2025-04-14T18:28:05.660156Z INFO ThreadId(01)
    omr: decode time: 305.5313ms
2025-04-14T18:28:05.660291Z INFO ThreadId(01)
    omr: All done
```

(E3): *[Parallelizability] [10 human-minutes + 10 compute-minutes]: Run the* InstantOMR *example with the number of threads equal to the number of payloads.*

Preparation: *Build the* InstantOMR *example.*

```
$ cargo +nightly build --package omr_core --
    example omr --features="nightly" --release
```

Execution: *Run* InstantOMR *example with 8 threads and 8 payloads.*

```
$ cargo +nightly run --package omr_core --
    example omr --features="nightly" --release
    -- --thread-count 8 --payload-count 8
```

Results: *The detect time of this example result is close to the latency got in the first Experiment* **E1**.

```
num threads: 8
all payloads count: 8
2025-12-17T10:37:53.263970Z DEBUG ThreadId(01)
    omr: Generating secret key pack...
2025-12-17T10:37:53.264594Z DEBUG ThreadId(01)
    omr: Generating sender and detector...
2025-12-17T10:37:53.875357Z DEBUG ThreadId(01)
    omr: Generating clues...
2025-12-17T10:37:53.877243Z INFO ThreadId(01)
    omr: gen clues time: 1.6446ms
2025-12-17T10:37:53.877342Z DEBUG ThreadId(01)
    omr: Generating payloads...
2025-12-17T10:37:53.877468Z INFO ThreadId(01)
    omr: gen payloads time: 35.7us
2025-12-17T10:37:53.877592Z DEBUG ThreadId(01)
    omr: Detecting...
[elapsed: 00:00:00] [########################]
    8/8 [eta: 00:00:00] [0s]
2025-12-17T10:37:54.158482Z DEBUG ThreadId(01)
    omr: Detect done
2025-12-17T10:37:54.158596Z INFO ThreadId(01)
    omr: detect time: 280.8241ms
2025-12-17T10:37:54.158686Z INFO ThreadId(01)
    omr: detect time per message: 35.103012ms
2025-12-17T10:37:54.159029Z INFO ThreadId(01)
    omr: encode indices times: 240.7us
2025-12-17T10:37:54.159150Z INFO ThreadId(01)
    omr: encode indices times per ciphertext:
    80.233us
2025-12-17T10:37:54.159574Z INFO ThreadId(01)
    omr: encode pertinent payloads time: 323.9
    us
```

```
2025-12-17T10:37:54.241155Z INFO ThreadId(01)
    omr: decode time: 81.4913ms
2025-12-17T10:37:54.241324Z INFO ThreadId(01)
    omr: All done
```

A.5 Notes on Reusability

We provide an alternative access on our GitHub repository https://github.com/xiangxiecrypto/tfhe-omr, which may contain updated code or instructions.

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/.