One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval

Alexandra Henzinger, Matthew M. Hong, and Henry Corrigan-Gibbs, *MIT;* Sarah Meiklejohn, *Google;* Vinod Vaikuntanathan, *MIT*

https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval

Alexandra Henzinger
MIT

Matthew M. Hong
MIT

Henry Corrigan-Gibbs
MIT

Sarah Meiklejohn
Google

Vinod Vaikuntanathan
MIT

A Artifact Appendix

A.1 Abstract

Our source code for our two new, high-throughput PIR schemes, SimplePIR and DoublePIR, is available under the MIT open-source license at https://github.com/ahenzinger/simplepir. SimplePIR and DoublePIR, including their extensions to support databases with long records and batch queries (cf. Sections 4.3 and 5.2), are implemented in roughly 1,400 lines of Go code, along with 200 lines of C (for the performance-critical matrix-multiplication routines). For each PIR scheme, the code implements the Setup, Query, Answer, and Recover routines. Our repository additionally contains a suite of correctness tests and performance benchmarks. To obtain our performance numbers, we run our benchmarks on an AWS EC2 `c5n.metal` instance.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The source code for SimplePIR and DoublePIR is available at https://github.com/ahenzinger/simplepir/tree/438b4590aceedf76c7588b03125dfc0db39e361f.

A.2.3 Hardware dependencies

We run our evaluation on an AWS EC2 `c5n.metal` instance running Ubuntu 22.04. However, it is possible to run the SimplePIR and DoublePIR code on any machine with an installation of Go and of a C compiler (see Section A.2.4), though this might require amending the command-line flags passed to the C compiler.

A.2.4 Software dependencies

Running SimplePIR and DoublePIR requires installations of Go and GCC. We additionally require Python, NumPy, and Matplotlib to generate our evaluation plots.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Instructions for installing the required dependencies are given in the `Setup` section of https://github.com/ahenzinger/simplepir/blob/main/README.md. Users should install Go (tested with version 1.19.1) and GCC (tested with version 11.2.0) to run SimplePIR and DoublePIR. Users should additionally install Python (tested with Python3), NumPy, and Matplotlib to generate our evaluation plots.

A.3.2 Basic Test

To run all SimplePIR and DoublePIR correctness tests, users should run the command

```
go test
```

in the `simplepir/pir` directory. The suite of correctness tests runs SimplePIR and DoublePIR on random databases of fixed dimensions and checks that the PIR outputs are correct. The test suite should take roughly 2.5 minutes to run.

This command should produce logging output, followed by this message to indicate that all tests have passed:

```
PASS
ok          github.com/ahenzinger/simplepir/pir
```

A.4 Evaluation workflow

A.4.1 Major Claims

Our paper makes the following claims:

(C1): SimplePIR performance. On a database 1 GB in size containing 2^{33} 1-bit entries, SimplePIR has:

1. a throughput of roughly 10 GB/s/core when running on an AWS EC2 `c5n.metal` instance,
2. 121 MB of offline download, and
3. 242 KB of online communication.

This is shown by experiment (E1), whose results are displayed in Table 8 in the body of our paper.

(C2): DoublePIR performance. On a database 1 GB in size containing 2^{33} 1-bit entries, DoublePIR has:

1. a throughput of roughly 7.4 GB/s/core when running on an AWS EC2 `c5n.metal` instance,
2. 16 MB of offline download, and
3. 345 KB of online communication.

This is shown by experiment (E2), whose results are displayed in Table 8 in the body of our paper.

(C3): Throughput with batching. SimplePIR and DoublePIR's throughput increases when the client makes batches of many queries at once. This is shown by experiment (E3), whose results are displayed in Figure 9 in the body of our paper.

(C4): Application evaluation. On a database 8 GB in size containing 2^{36} 1-bit entries, DoublePIR has:

1. a throughput of roughly 7 GB/s/core when running on an AWS EC2 `c5n.metal` instance,
2. 16 MB of offline download, and
3. 756 KB of online communication.

This is shown by experiment (E4), whose results are reported in Section 8.2 in the body of our paper.

A.4.2 Experiments

(E1): SimplePIR performance [5 compute-minutes]: This experiment benchmarks (1) the communication and (2) the throughput of SimplePIR, when running on a 1 GB database consisting of 2^{33} 1-bit entries.

How to: Run the command

```
LOG_N=33 D=1 go test -bench SimplePirSingle
↪ -timeout 0 -run=^$
```

from the `simplepir/pir` directory. The command will run SimplePIR 5 times on a database consisting of 2^{33} 1-bit entries, and print logging information including the communication and the per-core throughput of each run.

Results: For each run of SimplePIR, this experiment prints logging information that look as follows:

- `Offline download: 123572 KB`, indicating that SimplePIR's offline download consists of roughly 121 MB.
- `Online upload: 120.000000 KB`, indicating that SimplePIR's offline upload consists of 120 KB.
- `Rate: 10177.282855 MB/s`, indicating that SimplePIR's throughput was 10,177 MB/s/core.
- `Online download: 120.000000 KB`, indicating that SimplePIR's offline download consists of 120 KB.

(E2): DoublePIR performance [5 compute-minutes]: This experiment benchmarks (1) the communication and (2) the throughput of DoublePIR, when running on a 1 GB database consisting of 2^{33} 1-bit entries.

How to: Run the command

```
LOG_N=33 D=1 go test -bench DoublePirSingle
↪ -timeout 0 -run=^$
```

from the `simplepir/pir` directory. The command will run DoublePIR 5 times on a database consisting of 2^{33} 1-bit entries, and print logging information including the communication and the per-core throughput of each run. The logging results are interpreted the same way as in (E1).

(E3): Throughput with batching [1.5h compute-hours]: This experiment benchmarks SimplePIR's and DoublePIR's effective, per-core throughput, when run on batches of queries of increasing size, on a 1 GB database consisting of 2^{33} 1-bit entries.

How to: Run the command

```
go test -bench PirBatchLarge -timeout 0
↪ -run=^$
```

from the `simplepir/pir` directory. The command will run both SimplePIR and DoublePIR 5 times on a database consisting of 2^{33} 1-bit entries, with batch sizes ranging from 1 to 1024, and print logging information including the communication and the per-core throughput of each run. The logging results are interpreted the same way as in (E1).

Results: This command produces the output files `simple-batch.log` and `double-batch.log` in the `simplepir/pir` directory. From the `simplepir/eval` directory, run the command

```
python3 plot.py -p batch_tput -f
↪ ../pir/simple-batch.log
↪ ../pir/double-batch.log -n SimplePIR
↪ DoublePIR
```

This command plots SimplePIR and DoublePIR's effective, per-core throughput for various batch sizes, and writes this plot to the file `throughput_with_batching.pdf`. This command was used to generate Figure 9.

(E4): Application evaluation [40 compute-minutes]: This experiment benchmarks (1) the communication and (2)

the per-core throughput of DoublePIR, when running on an 8 GB database consisting of 2^{36} 1-bit entries.
How to: Run the command

```
LOG_N=36 D=1 go test -bench DoublePirSingle
↪ -timeout 0 -run=^$
```

from the `simplepir/pir` directory. The command will run DoublePIR 5 times on a database consisting of 2^{36} 1-bit entries, and print logging information including the communication and the per-core throughput of each run. The logging results are interpreted the same way as in (E1).

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.