Authenticated private information retrieval

Simone Colombo, *EPFL;* Kirill Nikitin, *Cornell Tech;* Henry Corrigan-Gibbs, *MIT;* David J. Wu, *UT Austin;* Bryan Ford, *EPFL*

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

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

USENIX'23 Artifact Appendix: Authenticated private information retrieval

Simone Colombo
EPFL

Kirill Nikitin
Cornell Tech

Henry Corrigan-Gibbs
MIT

David J. Wu
UT Austin

Bryan Ford
EPFL

1 Artifact Appendix

1.1 Abstract

The source code for our single- and multi-server authenticated-PIR schemes and the Keyd public-key server is available at https://github.com/dedis/apir-code under open-source license. The same repository contains unauthenticated-PIR schemes that we implemented as baselines for comparison; as single-server PIR baseline we use the original implementation of SimplePIR [HHCMV22]. Our implementation and the implementation of SimplePIR use C for the performance-critical functions. We perform all the experiments on machines equipped with two Intel Xeon E5-2680 v3 (Haswell) CPUs, each with 12 cores, 24 threads, and operating at 2.5 GHz, and 256 GB of RAM.

1.2 Description & Requirements

1.2.1 Security, privacy, and ethical concerns

None.

1.2.2 How to access

The source code for all the authenticated-PIR schemes, the classic-PIR schemes and Keyd under which this artifact evaluation was tested is available at https://github.com/dedis/apir-code/tree/af3202e3776d4cb880256372dd51613ee34532ba.

1.2.3 Hardware dependencies

We perform all the experiments on machines equipped with two Intel Xeon E5-2680 v3 (Haswell) CPUs, each with 12 cores, 24 threads, and operating at 2.5 GHz. Each machine has 256 GB of RAM, and runs Ubuntu 20.04 and Go 1.17.5. Machines are connected with 10 Gigabit Ethernet. In the experiments for the multi-server schemes and Keyd (Sections 7.1, 7.2 and 7.4), the client and the servers run on separate machines—the experiments use at most six machines. For single-server schemes we use a single machine that runs both client and server. However, it is possible to run the code, together with the accompanying tests, benchmarks and experiments, on any machine equipped the software dependencies listed in the next section.

1.2.4 Software dependencies

Running run the code requires Go (tested with Go 1.17.5 and 1.19.5) and a C compiler (tested with GCC 9.4.0).

Reproducing the evaluation results requires GNU Make, Screen, Python 3[1], Fabric, Tomli, Numpy and Matplotlib.

1.2.5 Benchmarks

None.

1.3 Set-up

1.3.1 Installation

Installation instructions are given in the `Setup` sections of https://github.com/si-co/vpir-code/blob/main/README.md and we report them here. To run the code in the repository install Go (tested with Go 1.17.5) and a C compiler (tested with GCC 9.4.0). To reproduce the evaluation results, install GNU Make, Screen, Python 3, Fabric, Numpy and Matplotlib.

1.3.2 Basic Test

To run all basic tests, users should clone the repository, and download the dump of the SKS PGP key directory using the command

```
bash scripts/download_sks_parsed.sh
```

in the repository's root directory.

To run the basic test, use the following command:

```
go test
```

in the repository's root directory. This command takes about six minutes to run and outputs the time taken by each test. If all the tests pass, the output ends as follows:

```
PASS
ok     github.com/si-co/apir-code
```

[1]The package `python-is-python3` might be needed.

1.4 Evaluation workflow

1.4.1 Major Claims

Our paper claims what follows.

Multi-server point queries (Section 7.1).

(C1): The maximum overhead for our multi-server authenticated-PIR scheme for point queries, in comparison with classic unathenticated PIR is 2.9× for user time and 1.8× for bandwidth. This is the outcome of experiment (E1), whose results are presented in Fig. 3.

(C2): The impact of the number of servers on our multi-server authenticated-PIR scheme for point queries is almost negligible for user time and imposes a linear increase for bandwidth. This is the result of experiment (E2), whose results are reported in Fig. 4 in the body of the paper.

(C3): The preprocessing cost for our multi-server authenticated PIR scheme for point queries is linear in the database size. This is the result of experiment (E3), whose results are reported in Fig. 8 in Appendix C.

Multi-server complex queries (Section 7.2).

(C4): The user time and bandwidth overheads of the authenticated-PIR schemes for complex queries against classic unauthenticated-PIR schemes are less than 1.1×. This is the outcome of experiment (E4), whose results are presented in Fig. 5.

Single-server point queries (Section 7.3).

(C5): The authenticated-PIR schemes from the decisional Diffie-Hellman assumption (DDH) and from the learning-with-errors assumption (LWE) have integrity error 2^{-128}. The DDH construction has a smaller digest, i.e., lower offline bandwidth, but has twice the online bandwidth of the LWE construction. The LWE construction is also faster (3-79×). The scheme with integrity amplification (LWE^+) has integrity error 2^{-64} but the same classic-PIR privacy as SimplePIR [HHCMV22]. LWE^+ is faster than LWE for the 1 KiB and 1 MiB databases, but slower (1.4×) for the 1 GiB database. SimplePIR is 30-100× faster than LWE^+. These results are the outcome of experiment (E5), whose results are presented in Fig. 6 in the body of the paper.

Application evaluation (Section 7.4).

(C6): For classic key look-ups we measure the wall-clock time needed to retrieve a PGP public-key with authenticated PIR, classic PIR without authentication, and by direct download. We measure 1.11 seconds for authenticated PIR, 1.10 seconds for unauthenticated PIR and 0.22 seconds for non-private direct look-up. This is the result of experiment (E6), whose results are discussed in Section 7.4 in the body of the paper.

(C7): To analyze the performance of Keyd in computing private statistics over keys, we measure user-perceived time and bandwidth of different predicate queries. For all the predicates, the user-perceived time and bandiwdth overheads of authenticated PIR are upper bounded by a factor of 1.05×. This is the outcome of experiment (E7), whose results are presented in Table 7 in the body of the paper.

1.4.2 Experiments

The experiments use at most six server machines (to run the client and servers) and an additional machine (that we call *local*) to manage the experiments. The local machine can be a commodity computer, since it is used only to run light scripts and gather results. Clone the repository on all the server machines and on the local machine.

(E1): [*15 human-minutes + 2 compute-hour*]: This experiment measures the user-time and bandwidth overheads of *two*-server authenticated-PIR schemes for point queries in comparison with unauthenticated PIR. This experiments uses three server machines: one client and two servers.

Preparation: Edit `simulations/multi/config.toml` on the *local* machine to indicate the IP address of the client machine and the IP addresses and ports of the two server machines. The default port numbers are safe to use.

Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e point
```

where `<username>` and `<password>` are the username and password for the servers, respectively, and `<path>` is the path of the repository's root on the servers.

Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e point
```

The command stores the figure in `simulations/multi/figures/point.eps`.

(E2): *[15 human-minutes + 18 compute-minutes]*: This experiment measures the impact of the number of servers on our multi-server authenticated-PIR schemes for point queries. This experiments uses six server machines: one client and five servers.

Preparation: Edit `simulations/multi/config.toml` on the *local* machine to indicate the IP address of the

client machine and the IP addresses and ports of the five server machines. The default port numbers are safe.
Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e point_multi
```

where `<username>`, `<password>` and `<path>` are as in experiment E1.
Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e point_multi
```

The command stores the figure in `simulations/multi/figures/multi.eps`.

(E3): *[5 human-minutes + 9 compute-minutes]*: This experiment measures the cost of preprocessing for our multi-server authenticated-PIR scheme for point queries. This experiment uses one server machine.
Preparation: Nothing.
Execution: Run the following commands from the repository's root on the *server* machine:

```
cd simulations
make preprocessing
```

Results: Run the following commands from the repository's root:

```
cd simulations
python plot.py -e preprocessing
```

The command stores the figure in `simulations/figures/preprocessing.eps`.

(E4): *[15 human-minutes + 36 compute-minutes]*: This experiment measures the user-time and bandwidth overheads of two-server authenticated-PIR schemes for complex queries in comparison with unauthenticated PIR. This experiments uses three server machines: one client and two servers.
Preparation: As in experiment E1. The file `simulations/multi/config.toml` on the *local* machine must list *only two* servers.
Execution: Run the following commands from the repository's root on the *local* machine:

```
cd simulations/multi
APIR_USER=<username>
APIR_PASSWORD=<password>
APIR_PATH=<path>
python simul.py -e predicate
```

where `<username>`, `<password>` and `<path>` are as in experiment E1.

Results: Run the following commands from the repository's root:

```
cd simulations/multi
python plot.py -e predicate
```

The command stores the figure in `simulations/multi/figures/complex_lines.eps`.

(E5): *[15 human-minutes + 21 compute-hour]*: This experiment measures the user-time and bandwidth overheads of single-server authenticated-PIR schemes for point queries in comparison with SimplePIR [HHCMV22]. This experiment uses one server machine.
Preparation: Nothing.
Execution: Run the following commands from the repository's root on the *server* machine:

```
cd simulations
make single
```

To evaluate SimplePIR, clone the following repository: https://github.com/si-co/simplepir. The code is the same as the original repository, but it runs the evaluation on the same database sizes as authenticated PIR and produces a compatible JSON file for the results. Run the following command (45 compute-minutes) from the repository's root on the *server* machine:

```
cd pir
go test -timeout 0 -run=PirSingle
```

Copy the file `simplePIR.json` (in the `pir` directory) in `simulation/results`.
Results: Run the following commands from the repository's root:

```
cd simulations
python plot.py -e single
```

The command stores the figure in `simulations/figures/single_bar.eps`.

(E6): *[20 human-minutes + 10 compute-minutes]*: This experiment measures the user-time needed download a PGP public-key with authenticated PIR for point queries, classic unauthenticated PIR for point queries and by direct download. This experiment uses three machines: one client and two servers.
Preparation: Download the dump of the SKS PGP key directory using the command

```
bash scripts/download_sks_parsed.sh
```

in the repository's root directory on the *two* servers. Set the IP addresses of the two servers in `simulations/real/real_client_pir.sh` and in `config.toml` (in the repository's root) on the client machine.
Execution: Run the following commands from the repository's root on the *first server*:

```
cd simulations/real
bash real_server_pir.sh 0
```

Similarly, on the *second server* run:

```
cd simulations/real
bash real_server_pir.sh 1
```

A server is running properly when it logs:

```
gRPC server started at <ip>:<port>
```

Once both servers started, run the following command on the *client machine*:

```
cd simulations/real
bash real_client_pir.sh
```

This command executes 30 look-ups with unauthenticated PIR and 30 with authenticated PIR. At the end, the client automatically shuts both servers down.

Results: Copy `simulations/results/stats_*` from the three machines (two servers and the client) on the *local* machine in the folder `/simulations/results`. Run the following commands:

```
cd simulations
python plot.py -e real
```

The command prints the results directly on the terminal.

(E7): *[20 human-minutes + 5 compute-hours]*: This experiment measures the user-time needed to compute statistics on the PGP public-keys with authenticated PIR for predicate queries and unauthenticated PIR. This experiment uses three machines: one client and two servers.

Preparation: As in Experiment E6, but set the IP addresses of the two servers in `simulations/real/real_client_fss.sh`.

Execution: Run the following commands from the repository's root on the *first server*:

```
cd simulations/real
bash real_server_fss.sh 0
```

Similarly, on the *second server* run:

```
cd simulations/real
bash real_server_fss.sh 1
```

For this experiment, it is not needed to wait for the servers to properly start. Run the following command on the *client machine*:

```
cd simulations/real
bash real_client_fss.sh
```

Results: Copy `simulations/results/stats_*` from the three machines (two servers and the client) on the *local* machine in the folder `/simulations/results`. Run the following commands:

```
cd simulations
python plot.py -e realcomplex
```

The command prints the results directly on the terminal. Table 7 has a different formatting, but values are the same as the one that the command prints on screen.

1.5 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/.