USENIX Security '26 Artifact Appendix: Do You Need a Receipt? Anonymous Credential Revocation at Continental Scale via Private Record Certification

Kasra EdalatNejad, Sebastian Faust, Jonas Hofmann, Philipp-Florens Lehwalder, Thomas Schneider
Technical University of Darmstadt, Germany
{edalat, schneider}@encrypto.cs.tu-darmstadt.de,
{sebastian.faust, jonas.hofmann1, philipp-florens.lehwalder}@tu-darmstadt.de

A Artifact Appendix

A.1 Abstract

We provide an artifact containing a proof-of-concept implementation of the Private Record Certification (PRC) protocol described in the paper. We also provide Docker files which automate the process of benchmarking our solution and generating the performance plots in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact is non-destructive and only collects performance measurement of its execution. There are no security, privacy, or ethical concerns.

A.2.2 How to access

The artifact is available on https://zenodo.org/records/20433787.

A.2.3 Hardware dependencies

This artifact requires an x86_64 CPU with AVX2 support. This includes most modern Intel and AMD CPUs, but excludes ARM-based platforms, including Apple Silicon.

Our benchmark evaluates up to 8 PRC servers. We measure single-core performance and restrict each server to one core. To simplify the process, we run all servers on the same machine with a simulated network. Therefore, we recommend running the full benchmark on a machine with at least 8 cores. If fewer cores are available, multiple servers will share cores, which may increase latency. The RAM footprint is low. A machine with 8 GiB of RAM should be sufficient for running the full 8-server benchmark.

A.2.4 Software dependencies

We provide a Docker container, based on Debian, to facilitate running the artifact. We require Docker to be installed on the device. We only provide instructions on how to run the container on a Linux machine, but it is possible to run it on different operating systems.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

To install the artifact, download it or clone the corresponding Git repository. The artifact requires Docker; if Docker is not already installed, follow the official installation instructions.[1] In the artifact's top-level directory, build the container:

```
sudo docker build -t prc .
```

A.3.2 Basic Test

This artifact is built in Rust, and we use `cargo test` for basic tests. You can use the following command to run all tests in the container. We expect that all tests should pass.

```
sudo docker run \
 --rm \
 -v "$(pwd)/measurements:/AnonCred/measurements" \
 prc \
 cargo test
```

A.4 Evaluation workflow

A.4.1 Major Claims

This artifact reproduces the server-side PRC measurements (cf. Figure 3). The benchmark directly measures latency, com-

[1] https://docs.docker.com/get-docker/

putation cost, and communication cost and estimates the operation's monetary cost based on the AWS price model.

(C1): Sub-second latency at billion-credential scale. PRC achieves sub-second latency even for databases with over one billion credentials. The latency is dominated by the simulated network delay. We also note that when RTT is 20 ms, the latency is under 100 ms. This is proven by the latency measures provided in `latency.pdf` plot as part of experiment (E1).

(C2): Linear server computation. The computation cost of PRC servers grows linearly with the number of credentials. This is proven by the server computation cost plotted in `comp_comm.pdf` generated in experiment (E1).

(C3): Sublinear server communication. The communication cost of PRC servers grows sublinearly with the number of credentials; more precisely, it grows as $O(\sqrt{N})$. This is proven by server communication cost plotted in `comp_comm.pdf` generated in experiment (E1).

(C4): Efficient monetary cost at scale. The monetary cost grows linearly with the number of credentials, and the protocol remains efficient even when processing millions of PRC queries. The estimated monetary cost is computed from measured resource consumption using AWS pricing. Our pricing model is as follows: one CPU core, less than 2 GiB RAM, $0.04 per compute hour, free inbound transfer, and $0.09 per GiB outbound. Our cost efficiency is proven by the operation cost plotted in `costs.pdf` generated in experiment (E1).

The paper also makes the following client-side claims, which are not directly shown in the produced figures.

(C5): Client cost is independent of the number of revocations. In our protocol, revoking a credential only requires changing a value in the servers' database. There is no corresponding client-side update, and client queries are agnostic to the changes in the servers' database. The code shows that any change in revocation status is fully server-side, with no impact or notification to clients.

(C6): Client cost is logarithmic to the number of credentials. In theory, the client cost is logarithmic in the number of credentials because the record index size grows with the database size. In practice, we represent indices as a single integer where a `u32` supports up to four billion records. Therefore, within the evaluated range, the client cost is effectively independent of the number of credentials.

The paper compares the cost of supporting anonymous-credential revocation using our PRC construction against a naive PIR-based approach and ALLOSAUR. For the naive PIR baseline, the cost corresponds to performing a single PIR query over a database of n records, each of size 96 B. For ALLOSAUR, we use the implementation as provided, without modification.

Comparison with related work Because these comparisons evaluate unmodified external implementations—Hintless PIR[2], DPF-based PIR[3], and ALLOSAUR[4]—we do not include their source code in this artifact. A database with one billion 96 B entries requires 96 GB of storage, which is beyond the scalability of existing PIR implementations. We therefore extrapolate costs for settings in which prior systems fail to run.
We describe how we ran the external implementations, the raw measurements we obtained, and the extrapolation method used beyond their scalability limits. Since the external projects already provide their own installation and usage instructions, we do not include detailed setup instructions or automation for those packages.

A.4.2 Experiments

(E1): [human-interaction: minimal, compute-time: 3:30 min, peak memory: 1 GiB]: Run the automated evaluation.
How to: The evaluation is fully automated. After installing Docker and building the image per set-up instructions, run the following command:

```
sudo docker run \
 --cap-add=NET_ADMIN \
 --rm \
 -v "$(pwd)/measurements:/AnonCred/measurements" \
 prc \
 bash /AnonCred/bench_run_all.sh
```

Results: This script runs PRC protocols with $\{2,4,8\}$ PRC servers on a single machine with a simulated network—setting an RTT of $\{20\,\text{ms}, 40\,\text{ms}, 80\,\text{ms}\}$ via `tc` command—and measures the latency, server computation, and communication cost of running a PRC query based on the number of records held by the server. We run each experiment 5 times, record the performance measurement, and plot the costs. After the benchmark finishes, the following plots `latency.pdf`, `costs.pdf`, and `comp_comm.pdf` will be generated in the `measurements/` directory. These three plots correspond to Figure 3 in the paper and support our claims.

The PRC measurements reported in the paper are measure on a system running Debian 13 with an AMD Ryzen AI 7 PRO 350 CPU and 64 GB RAM.

A.5 Notes on Reusability

We provide detailed instructions on how to build the artifact from the source, manually run the server with customizable parameters, and describe the artifact structure and subcomponents that may be of independent interest, such as SIMD-aligned bit-arrays, our custom MPC engine, conversions, and PIR construction in the artifact readme. If you are interested in using/reusing our prototype, please refer to the latest version of the repository.

[2] https://github.com/google/hintless_pir
[3] https://github.com/google/distributed_point_functions
[4] https://github.com/sam-jaques/allosaurust

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/.