Orca: Blocklisting in Sender-Anonymous Messaging

Nirvan Tyagi and Julia Len, *Cornell University;* Ian Miers, *University of Maryland;* Thomas Ristenpart, *Cornell Tech*

https://www.usenix.org/conference/usenixsecurity22/presentation/tyagi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

B Artifact Appendix

B.1 Abstract

Our artifact contains source files of the Orca blocklisting protocol as a library in Rust. The cryptographic protocol is built on top of the open-source `arkworks` library for pairing-based cryptography. The implementation consists of three major parts: (1) an implementation of the Chase et al. algebraic MAC protocol, (2) an implementation of the Orca group signature, and (3) an implementation of the Orca one-time-use token protocol. The artifact also includes two benchmarks for reproducing the performance numbers reported on. These benchmarks can be easily run on any machine that can compile Rust from source, though we report performance numbers from running on high-memory AWS machines (for the server) and mobile devices (for the client). The artifact does not include source files for the griefing attack and battery-drain experiments against Signal, as they are potentially harmful and are not core to our work's claimed contribution.

B.2 Artifact check-list (meta-information)

- **Algorithm:** The Orca blocklisting protocol including group signature and one-time-use tokens.
- **Compilation:** Benchmarks are built from source using the Rust compiler.
- **Run-time environment:** Our artifact was run on a `c5.12xlarge` AWS EC2 virtual machine with 24 cores and 96 GB of memory running Ubuntu Server 20.04 LTS, as well as on a mobile device running Android 9.
- **Hardware:** The mobile microbenchmarks were run on a Google Pixel 2 device. The server throughput benchmark requires at least 64 GB, though comparable results can be reproduced with less memory.
- **Execution:** The microbenchmarks run in less than 5 minutes. The server throughput benchmark runs in under 2 hours on our test AWS machine.
- **Security, privacy, and ethical concerns:** We do not provide the source files for the griefing attack and battery-draining experiments.
- **Output:** The benchmarks produce summarized performance outputs printed to the terminal.
- **Experiments:** There are two benchmarks: (1) microbenchmarks for measuring the performance of the cryptographic primitives used in Orca, and (2) macrobenchmark for measuring server throughput of requests.
- **How much time is needed to prepare workflow (approximately)?:** The benchmark binaries are built from source in under 5 minutes. Setting up the AWS machine and/or the mobile device may take additional time.
- **Publicly available?:** The latest version of the library is available at `https://github.com/nirvantyagi/orca`. The version that underwent artifact review is marked with tag `usenix-sec22-ae`.

B.3 Description

B.3.1 How To Access

The latest version of the library is available at `https://github.com/nirvantyagi/orca`. The version that underwent artifact review is marked with tag `usenix-sec22-ae`.

B.3.2 Hardware Dependencies

Our artifact was run on a `c5.12xlarge` AWS EC2 virtual machine with 24 cores and 96 GB of memory running Ubuntu Server 20.04 LTS. The server throughput benchmark requires at least 64 GB, though comparable results can be reproduced with less memory. The mobile microbenchmarks were run on a Google Pixel 2 device running Android 9.

B.3.3 Software Dependencies

Full instructions for building from source are provided on the project README. All dependencies are readily available through the Rust package manager and binaries can be built from source in under 5 minutes.

B.3.4 Security, Privacy, and Ethical Concerns

We do not provide the source files for the griefing attack and battery-draining experiments.

B.4 Installation

The setup consists of installing Rust and compiling the benchmark binaries from source. Compiling and running the microbenchmarks on a mobile device requires additional installation of the Android Native Development Kit (NDK) and related Rust toolchains. The macrobenchmark for server throughput additionally requires installing and running a Redis server locally. Detailed installation instructions are given on the README available at `https://github.com/nirvantyagi/orca`.

B.5 Evaluation and Expected Results

There are two benchmark binaries that we report results on. The first is the microbenchmarks binary that is used to populate Figure 5. The platform and desktop client user columns are given from running the microbenchmark binary on a single core of the specified AWS machine. The mobile client user column is given from running the microbenchmark on the specified mobile device.

The second benchmark binary measures server throughput and is used to populate Figure 6. The reported numbers are based on experiments setting benchmark parameters of 200 requests for a blocklist size of 100, a strikelist size of 1400, and one million users, while varying the number of cores. This setup requires 64 GB of memory, however, the number of users can be reduced (e.g., to 200) to reproduce similar results without large memory requirements.

Detailed evaluation instructions are given on the README available at https://github.com/nirvantyagi/orca.

B.6 Experiment Customization

The benchmark source code is available and can be customized beyond the exisiting parameterization.

B.7 Notes

The cryptographic code has not been reviewed; it serves as a research prototype and is not suitable for deployment. If any bugs are discovered, please raise an issue on Github or send an email to the authors.