Secure Account Recovery for a Privacy-Preserving Web Service

Ryan Little, *Boston University;* Lucy Qin, *Georgetown University;*
Mayank Varia, *Boston University*

https://www.usenix.org/conference/usenixsecurity24/presentation/little

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: Secure Account Recovery for a Privacy-Preserving Web Service

Ryan Little
Boston University

Lucy Qin
Georgetown University

Mayank Varia
Boston University

A Artifact Appendix

A.1 Abstract

Account recovery—the ability for users to regain access to their accounts after losing their password—is a near-ubiquitous feature of account-based web service. In a typical account recovery system, the server maintains a list of user email addresses to contact users who forgot their password. Our paper proposes a new cryptographic protocol for account recovery with improved privacy. On the surface, our protocol appears to follow the typical workflow: the user types in their email address, receives an email containing a one-time link, and answers some security questions. But on the backend, our protocol removes the need for the server to store identifiable user information.

The core building block of our protocol is a novel primitive that combines the characteristics of an oblivious pseudorandom function (OPRF) and a partially-oblivious pseudorandom function (pOPRF). This primitive, which we call a kaleidoscopic partially-oblivious PRF (K-pop), is an interactive procedure between two parties (a client and a server) that allows the client to learn an output of a pseudorandom function that takes in with two inputs, x_{kal} and x_{priv}. The K-pop can either be evaluated in OPRF mode, in which case both x_{kal} and x_{priv} are secret inputs chosen by the client, or in pOPRF mode, in which case x_{priv} is chosen by the server and x_{kal} is a secret input chosen by the client. Both OPRF mode and pOPRF mode compute the same function.

Our artifact is an implementation of the K-pop primitive, written in Sage. It contains code allowing a client and server to evaluate the K-pop in both OPRF and pOPRF mode. The implementation supports five different "ciphersuites", which specify an elliptic curve group and hash function. The artifact contains a correctness test that checks that the K-pop computes the same function whether it is in OPRF mode or pOPRF mode, as well as two benchmarking tests that can re-produce the timing results in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is accessible at https://github.com/ryanjlittle/kpop-oprf/tree/ae6c354d84ed3d74d47d25cf31484c7d6f9edaf4.

A.2.3 Hardware dependencies

Our multiprocessing benchmark test requires a CPU with (at least) 4 cores. The benchmarks in our paper were produced on a Lenovo ThinkPad P15s running Manjaro Linux with a 4-core 1.6GHz Intel Core i5-10210U CPU and 16GB RAM.

A.2.4 Software dependencies

Two software dependencies are required: A Python 3 installation (we used version 3.12.3), and a SageMath installation (we used version 10.3).

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

1. Install Python. The latest source releases are available at https://www.python.org/downloads/.
2. Install SageMath. On macOS with Homebrew, Sage can be installed with the command `brew install --cask sage`. On Arch-based Linux systems, it can be installed with `sudo pacman -S sagemath`. You can check that Sage was correctly installed by running `sage` from the command line. For other operating systems and detailed installation instructions, refer to https://doc.sagemath.org/html/en/installation/. Note that a development version of Sage is required.
3. Clone the repository. The repo contains a submodule. To ensure it gets cloned correctly, clone the repo with `git clone --recurse-submodules \ https://github.com/ryanjlittle/kpop-oprf.git`. Alternatively, clone the repository normally and run

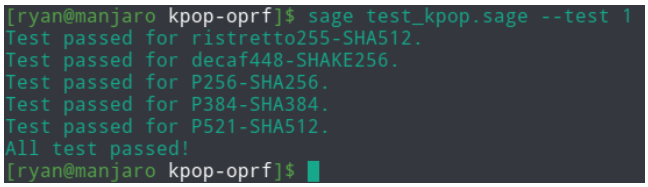

Figure 1: Console output of a successful basic test

`git submodule update --init` from inside the main directory.

4. Build the source code by running `make` from inside the main directory.

A.3.2 Basic Test

After following the installation and building instructions, the correctness of the implementation can be checked by running `sage test_kpop.sage --test 1`. This runs the K-pop in both OPRF mode and pOPRF mode on a randomly chosen input for all five supported ciphersuites. It checks that the K-pop yields the same result whether it is evaluated in OPRF mode or pOPRF mode. A successful run will complete in a few seconds and yield the output shown in figure A.3.2. The final input argument (the number 1) fixes the number of random tests. For a more thorough correctness test, this argument can be set to a higher number.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Single-core K-pop performance results, shown in figure 9 of the paper. This figure shows the average amount of server and client time to run the K-pop in both OPRF mode and pOPRF mode over five different ciphersuites. This experiment is described in section 5.1 of the paper, and can be reproduced by the experiment (E1).

(C2): Multi-core K-pop performance results, shown in figure 10 of the paper. These results measure the performance of the server-side K-pop work in a multi-processing setting, with $P \in \{1,2,4\}$ cores. Our results show that a single server can scale to handle many clients simultaneously by running multiple processes in parallel. Our tests show that a four-core server has amortized performance that is 2-3 times better than a single-core server. The experiment is described in section 5.2 of the paper, and can be reproduced with the experiment (E2).

A.4.2 Experiments

(E1): [Single-core performance results] [5 human-minutes + 5 compute-minutes]: This experiment simulates a K-pop interaction between a client and server (both instantiated locally) in both OPRF mode and pOPRF mode, for each of the five supported ciphersuites (ristretto255-SHA512, decaf448-SHAKE256, P256-SHA256, and P521-SHA512). It runs 1000 tests for each combination of mode and ciphersuite, and takes the average client and server time for each.
Preparation: Install and build the artifact, following the instructions in section A.3.1.
Execution: From the main directory, run `sage test_kpop.sage --figure 1000`. This will take around 5 minutes to complete.
Results: The experiment will produce a graph and console outputs. The graph should resemble figure 9 in the paper. Exact time values will of course vary depending on your hardware, but the relative heights of the bars should look similar. The console output prints the mean value and standard error for each set of measurements.

(E2): [Multi-core performance results] [5 human-minutes + 5 compute-minutes]: This experiment simulates a batch of 512 clients simultaneously interacting with one server that delegates each client to one of P parallel processes, for $P \in \{1,2,4\}$. The work is evenly split such that each process handles $512/P$ clients. This is done for each combination of ciphersuite and OPRF/pOPRF mode.
Preparation: Install and build the artifact, following the instructions in section A.3.1.
Execution: From the main directory, run `sage test_kpop.sage --benchmark 512`. This will take around 5 minutes to complete.
Results: The experiment will produce console output showing the average server evaluation time for each combination of ciphersuite/mode/P combination. These values should be similar to the results in figure 10.

A.5 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/usenixsec2024/.