

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SIGMA-RS: A Modular Approach for Keyed-Verification Anonymous Credentials

Michele Orrù, *CNRS;* Lindsey Tulloch, *Tor Project, University of Waterloo;* Victor Snyder-Graf, *RISC Zero;* Ian Goldberg, *University of Waterloo*

https://www.usenix.org/conference/usenixsecurity26/presentation/orru

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: SIGMA-RS: A Modular Approach for Keyed-Verification Anonymous Credentials

Michele Orrù
CNRS

Lindsey Tulloch
Tor Project, University of Waterloo

Victor Snyder-Graf
RISC Zero

Ian Goldberg
University of Waterloo

A Artifact Appendix

A.1 Abstract

We introduce a new software stack in Rust aimed at simplifying constructions and deployments of protocols based on modern anonymous credential systems.

The stack, called SIGMA-RS, through its layered design, abstracts cryptographic complexity while remaining flexible enough to support a range of credential schemes, proofs, and access policies. It emphasizes misuse resistance via type safety, domain separation, and prover-state discipline, and supports side-channel-aware constant-time strategies.

We evaluate practicality through re-implementations of Tor's Lox bridge distribution protocols and of user authentication in the Open Observatory for Network Interference (OONI).

The artifact will allow you to run all of the self-tests in our SIGMA-RS stack and the Lox and OONI applications, and to reproduce Tables 2, 3, and 5 in the paper.

A.2 Description & Requirements

This repository contains the code artifact for SIGMA-RS, a Rust software stack for implementing protocols based on keyed-verification anonymous credentials (KVAC).

The SIGMA-RS stack consists of the following components (listed along with the versions pinned in this artifact):

- spongefish tag v0.6.1
- sigma-proofs tag v0.3.1
- sigma-compiler tag 0.2.2
- cmz tag 0.2.1

This artifact also evaluates two sample applications using this stack, described in the paper:

- application-ooni tag artifact-v0.4
- application-lox tag lox-artifact

We also include for comparison a version of application-lox that uses an update of the older zkp crate, instead of our SIGMA-RS stack:

- application-lox-zkp tag lox-artifact-zkp

Artifact structure. The directories in this repository are as follows:

Scripts: Useful scripts for building a docker image for this artifact, and running tests therein

patches: Patches to the Cargo.toml file for our SIGMA-RS collection of crates, to force them to use the local copies of each other as dependencies, rather than using the published versions from crates.io. These patches are applied automatically by the `build-docker` script.

A.2.1 Security, privacy, and ethical concerns

There is no risk to the evaluators.

A.2.2 How to access

Our artifact is available at `https://git-crysp.uwaterloo.ca/SigmaProtocol/sigma-rs-artifact` with an archival copy at `https://doi.org/10.5281/zenodo.20190651`.

A.2.3 Hardware dependencies

Most of the results in this artifact can be reproduced on any machine that can run a recent Linux distribution (64-bit x86 or ARM). For best results (more closely matching those in the paper), use a CPU with AVX512 support. However, the artifact will still work, albeit with slower results, if you have only AVX2 or no AVX support at all.

To execute the OONI iOS benchmarks (the "iOS" column in Figure 3 in the paper), you will need a Mac host with Xcode and rustup installed, and an iOS device on which you can install apps you compile yourself.

A.2.4 Software dependencies

We assume your host runs Ubuntu 24.04, but any similar system should be fine. You will need installed on the host:

- git
- Either docker or podman. The scripts will auto-detect which of docker or podman you have. The *names* of the scripts contain the word "docker", even if they end up using podman instead.
- A wasm-capable web browser, which is pretty much any modern browser. You cannot use Tor browser, since you'll be connecting to a localhost web server, which you cannot do over the Tor network.
- As noted in Section A.2.3, the Mac host used to execute the OONI iOS benchmarks will need to have Xcode and rustup installed.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

After downloading or cloning the repository, build a docker image with:

```
./Scripts/build-docker
```

On a recentish laptop, this image should take around 10 minutes to build. The resulting image is about 9 GB.

A.3.2 Basic Test

To ensure everything has built properly, you should run the unit tests within the docker with:

```
Scripts/run-docker Scripts/run_all_tests
```

This should take less than 30 seconds to run.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The timings of our modular SIGMA-RS re-implementation of the Lox protocols are reasonable for modern web applications such as Tor for each of the protocols in both the native and wasm environments. They are also comparable to those of the original Lox implementation using the zkp library. This claim is supported by Experiment (E1), whose results are reported in Section 8.1 of the paper (Table 2).

(C2): The request and response sizes of our modular SIGMA-RS re-implementation of the Lox protocols are comparable to those of the original Lox implementation using the zkp library. This claim is supported by Experiment (E1), whose results are reported in Appendix D of the paper (Table 5).

(C3): Our implementation of the OONI protocols using our SIGMA-RS stack shows protocol operations completing within a few milliseconds on both the client and server, with comparable performance across the native and iOS platforms and no significant overhead introduced by mobile execution. This claim is supported by Experiments (E2) and (E3), whose results are reported in Section 8.2 of the paper (Table 3).

A.4.2 Experiments

(E1): *[Lox native and wasm benchmarks] [a few minutes]: This experiment runs both the original implementation of the Lox protocols using the zkp library, as well as our re-implementation using our* SIGMA-RS *stack. Both implementations are run in both native code and wasm environments. The time to perform the protocols and the sizes of the request and response messages are reported. This experiment supports Claims (C1) and (C2).*

Preparation: Have built the docker (or podman) image following Section A.3.1 above. Open a wasm-capable browser (running on the same machine that is running the experiment); be ready to load a URL with it. See Section A.2.4 above for more information about the wasm-capable browser.

Execution: To run the Lox native and wasm benchmarks (Tables 2 and 5 of the paper):

```
Scripts/run-docker Scripts/run_lox_benches
```

When the wasm benchmarks are run (twice: once for the new SIGMA-RS version of Lox, and once for the original zkp version), the script will prompt you with a URL to load in the wasm-capable web browser. Do so when prompted each of the two times.

Results: The output of the script will end with the data tables corresponding to Tables 2 and 5 in the paper. The values in Table 2 are times in milliseconds (with stddevs in parens). The values in Table 5 are sizes in bytes.

(E2): *[OONI native benchmarks] [30 seconds]: This experiment runs our* SIGMA-RS *implementation of the OONI protocols in the native environment. The times to perform the protocols are reported. This experiment supports Claim (C3).*

Preparation: Have built the docker (or podman) image following Section A.3.1 above.

Execution: To run the OONI native benchmarks (the "native" columns in Table 3):

```
Scripts/run-docker Scripts/run_ooni_benches
```

Results: The output of the script will be the data tables corresponding to the "native" columns of Table 3 in the paper. The values are times in milliseconds (means and stddevs).

(E3): *[OONI iOS benchmarks] [2 minutes combined for preparation and execution]: This experiment runs our* SIGMA-RS *implementation of the OONI protocols in the iOS environment. The times to perform the protocols are reported. This experiment supports Claim (C3).*

Preparation: As mentioned in Section A.2.3 above, this experiment must be run on a Mac host with Xcode and rustup installed, and an iOS device on which you can install apps you compile yourself.

If you downloaded the artifact from zenodo, or if you built the docker image following Section A.3.1 above, then you will have an `application-ooni` directory. Otherwise (for example, if you do not have docker on your Mac host, and you cloned the artifact repository from git-crysp.uwaterloo.ca), you can clone the dependency repositories with:

```
Scripts/clone-repos
```

In either case, once you have the `application-ooni` directory:

```
cd application-ooni
rustup target add aarch64-apple-ios \
   aarch64-apple-ios-sim x86_64-apple-ios
./ios/build-ffi.sh
```

This writes the file `ios/OoniAuthBindings/OoniAuthFFI.xcframework`.

Open `ios/OoniAuthApp.xcodeproj` in Xcode, select the `OoniAuthApp` target, build and then run, selecting your iOS device as the destination.

If the build fails with `xcodebuild ... requires Xcode` or a missing iOS SDK:

```
sudo xcode-select -s \
 /Applications/Xcode.app/Contents/Developer
```

Execution: Open the app, which has a single button. Press the button.

Results: The output of the app will be the data corresponding to the "iOS" column of Table 3 in the paper. The values are times in milliseconds (means and stddevs).

A.5 Notes on Reusability

For instructions on using the SIGMA-RS stack to implement your own KVAC protocols, see README.md in the cmz repository.

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/.