

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC

Alexander Bienstock, Leo de Castro, Daniel Escudero, Antigoni Polychroniadou, and Akira Takahashi, *J.P. Morgan AlgoCRYPT CoE and J.P. Morgan AI Research*

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

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: Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC

Alexander Bienstock Leo de Castro Daniel Escudero
Antigoni Polychroniadou Akira Takahashi

JPMC AlgoCRYPT CoE
{firstname}.{lastname}@jpmchase.com

March 18, 2026

A Artifact Appendix

A.1 Abstract

This artifact is a C++ implementation of the online phase of our protocol. The implementation is built atop the Secure Computation Library[1], which implements the channel & network logic for MPC party communication. We provide a simple benchmark file to time the online computation of our protocol for various number of parties at all three ML-DSA security levels. The results of this benchmark are given in Table 6 on page 14 of our paper.

A.2 Description & Requirements

This artifact is a self-contained C++ library that should be compiled to run the benchmark binary. We have tested this library on several version of Linux, including Ubuntu and the AWS EC2 default Linux OS. We have preserved the original `README.md` file from the SCL library, and the build instructions are identical.

Dependencies. There are no major 3rd-party dependencies for this library aside from a compiler, CMake, and some standard C++ libraries. Aside from the basic `build-essentials` like the `make` command, the library should only require

- `g++-13`
- `gmp`
- `catch2`

to compile. We are happy to assist with any issues installing dependencies.

Building the library. To facilitate building the library in the various modes, we have included scripts that run the entire build process. For example, to build the library in "debug" mode, simply run `bash build_debug.bash` from the top directory. This builds the library from scratch and creates a new `build` folder where the binaries are stored. In `debug` mode, the library will perform many correctness checks as it evaluates the test circuits.

Testing the code. To run the complete unit-test suite for this library, simply build the code in `debug` mode and run the binary `./build/test/scl_test`. There is also a binary located in `./build/test_norm_check` that evaluates specific tests for the complete online phase circuit.

Running the benchmark. To run the benchmark of the online phase of our protocol, we recommend to first build the library in `release` mode by running `bash build_release.bash` from the top directory. The benchmark script will still run in `debug` mode, but it will be much slower. Once the library is built, run the binary located at `./build/bench_norm_check`. This binary may take a few minutes to fully complete, and it produces all the data in table 6 in our paper.

A.2.1 Security, privacy, and ethical concerns

This artifact is simply a self-contained C++ program that does not require any special permissions to compile or run. There is no external data used to generate this program; it is simply an implementation of the algorithms described in the paper. We are not aware of any security, privacy, or ethics concerns with this artifact.

A.2.2 How to access

The code is available on Zenodo at this url: https://doi.org/10.5281/zenodo.17888654

ACCESSING THE ARTIFACT: This code was written at JP Morgan Chase, and there are procedures we must follow before publicly releasing any piece of software (even

[1] https://github.com/anderspkd/secure-computation-library

research software) from a regulated financial institution. We have received permission from the AEC chairs to keep the Zenodo files restricted for now. We will make the files public as soon as we have the necessary approvals. In the meantime, please feel free to request access to the Zenodo file. We will accept these requests promptly, and we appreciate your understanding. In order to maintain anonymity, we will accept any request we receive that mentions the USENIX artifact evaluation in the message; feel free to use an obfuscated/anonymized email and a fake name in the request.

A.2.3 Hardware dependencies

None. This artifact should not require any special hardware beyond a normal CPU that can support the standard x86 instruction set.

A.2.4 Software dependencies

The only dependencies our code requires are those inherited from the SCL library as well as g++-13. We give the commands to install each dependency on a Ubuntu Linux OS.

- `gmp`: `sudo apt install libgmp-dev`
- `catch2`: `sudo apt install libcatch2-dev`
- `g++-13`:
  1. `sudo apt install -y software-properties-common`
  2. `sudo add-apt-repository ppa:ubuntu-toolchain-r/test`
  3. `sudo apt update`
  4. `sudo apt install g++-13`

A.2.5 Benchmarks

None. All of the benchmarks in our work are self-contained timings of the online phase of our protocol.

A.3 Set-up

A.3.1 Installation

This library does not need any special permissions to build or run. To build the source code, simply unzip the file stored in the Zenodo entry, install the dependencies, and run the build scripts. We have provided the scripts `build_debug.bash` and `build_release.bash` to facilitate this process. Simple run these scripts from the top directory of the library to build the code. The libraries are placed in the `./build` directory.

A.3.2 Basic Test

To run a simple test, first build the library in `debug` mode by running `bash build_debug.bash`. The unit-tests for this library can be run with the binary `./build/scl_test`. Tests for the online phase of our protocol can be run with `./build/test_norm_check`.

A.4 Evaluation workflow

To run the evaluation, we recommend building the library in `release` mode by running the command `bash build_release.bash` run from the top directory. The benchmark binary can then be run with the command `./bench_norm_check`.

A.4.1 Major Claims

(C1): The script benchmarks the online phase for our protocol, evaluating both the batched comparison function and the batched OR protocol of the online phase, with a batch size matching all three security levels of ML-DSA.

A.4.2 Experiments

(E1): *[Online Phase Benchmark] [2 human-minutes]:* Run the binary `bench_norm_check` to obtain the data in table 6 of our paper. Our benchmarks were run on a single thread of a machine with 32 GB of RAM with an Intel i7 chip running at 2.5 GHz.

A.5 Notes on Reusability

N/A

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/.