USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

MAESTRO: Multi-Party AES Using Lookup Tables

Hiraku Morita, *Aarhus University and University of Copenhagen;* Erik Pohle, *COSIC, KU Leuven;* Kunihiko Sadakane, *The University of Tokyo;* Peter Scholl, *Aarhus University;* Kazunari Tozawa, *The University of Tokyo;* Daniel Tschudi, *Concordium and Eastern Switzerland University of Applied Sciences* (*OST*)

https://www.usenix.org/conference/usenixsecurity25/presentation/morita

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: MAESTRO: Multi-Party AES Using Lookup Tables

Hiraku Morita
Aarhus University
University of Copenhagen

Erik Pohle
COSIC, KU Leuven

Kunihiko Sadakane
The University of Tokyo

Peter Scholl
Aarhus University

Kazunari Tozawa
The University of Tokyo

Daniel Tschudi
Concordium
Eastern Switzerland University of Applied Sciences (OST)

A Artifact Appendix

A.1 Abstract

Secure multi-party computation (MPC) enables multiple distrusting parties to jointly compute a function while keeping their inputs private. Computing the AES block cipher in MPC, where the key and/or the input are secret-shared among the parties is important for various applications, particularly threshold cryptography.

In this work, we propose a family of dedicated, high-performance MPC protocols to compute the non-linear S-box part of AES in the honest majority setting. Our protocols come in both semi-honest and maliciously secure variants. Our implementation of the MPC protocols for three parties will be the main artifact.

Our protocols have different trade-offs, such as having a similar round complexity as previous state-of-the-art by Chida et al. [WAHC'18] but 37% lower bandwidth costs, or having 27% fewer rounds and 16% lower bandwidth costs. An experimental evaluation in various network conditions using three party replicated secret sharing shows improvements in throughput between 28% and 71% in the semi-honest setting. For malicious security, we improve throughput by 319% to 384% in LAN and by 717% in WAN due to sublinear batch verification.

A.2 Description & Requirements

The artifact is software, given as source code, and the raw benchmark data. The software part implements all protocols that were evaluated experimentally and the raw benchmark data is the raw resulting runtime/communication cost, etc. when running the software on our servers.

A.2.1 Security, privacy, and ethical concerns

The source code provided does not pose any risk. There is a minimal risk of a supply chain attack as the code uses external libraries. We are not aware of any privacy or ethical concerns arising from benchmarking or testing our code.

A.2.2 How to access

The source code and raw benchmark data can be found in https://github.com/KULeuven-COSIC/maestro, or archived in https://doi.org/10.5281/zenodo.14719154. The `README` file in the root directory provides instructions on how to use the artifact and how to parse the raw benchmark data.

A.2.3 Hardware dependencies

Evaluation of this artifact does not require special hardware. Benchmarking requires three servers/computers with `x86` or `AArch64` architecture offering CLMUL support (see Section A.3.2 on how to test for this). The throughput and latency of the network connecting the servers are specific to the different benchmarks. The settings range from "WAN network" (50 Mbits/s and 100 ms round-trip time) to "LAN network" (10 Gbit/s and almost no latency). More details are found in the `README` file.

The benchmark results in the paper were created on three separate servers with 16 CPU cores and 128GB RAM. To validate our claims, in particular (C1) for functionality, it is not necessary to have access to similar hardware. To validate (C2) or (C3), three machines with 4 to 8 cores, and 8GB to 16GB RAM should suffice to run the protocols with smaller batch sizes.

A.2.4 Software dependencies

Our software was developed on Linux and MacOS using the Rust programming language. All dependencies are freely available online and are described in the `README` file.

A.2.5 Benchmarks

The raw data from the experiments reported in the paper can be found in the `benchmark-data` folder. We consider throughput and latency benchmarks. The details of the data format are described in the `README` file mention above.

Throughput Benchmarks

- `benchmark-data/10Gbit` contains data of all protocols in the 10 Gbit/s network with batch sizes 50000, 100000 and 250000.
- `benchmark-data/1Gbit` contains data of all protocols in the 1 Gbit/s network with batch sizes 50000, 100000 and 250000.
- `benchmark-data/200Mbps-15msRTT` contains data of all protocols in the 200 Mbit/s with 15 ms round trip time network with batch sizes 50000, 100000 and 150000.
- `benchmark-data/100Mbps-30msRTT` contains data of all protocols in the 100 Mbit/s with 30 ms round trip time network with batch sizes 10000, 50000 and 100000.
- `benchmark-data/50Mbps-100msrtt` contains data of all protocols in the WAN network (50 Mbit/s with 100 ms round trip time) with batch sizes 10000, 50000 and 100000.

Latency Benchmarks

- `benchmark-data/10Gbit-latency` contains data for 1 AES block in the 10 Gbit/s network,
- `benchmark-data/1Gbit-latency` contains data for 1 AES block in the 1 Gbit/s network,
- `benchmark-data/200Mbps-15msRTT-latency` contains data for 1 AES block in the 200 Mbit/s with 15 ms round trip time,
- `benchmark-data/100Mbps-30msRTT-latency` contains data for 1 AES block in the 100 Mbit/s with 30 ms round trip time,
- `benchmark-data/50Mbps-100msrtt-latency` contains data for 1 AES block in the WAN network.

A.3 Set-up

All information on installation, tests, and evaluation workflow are also found in the `README` file in the root directory of the artifact.

A.3.1 Installation

1. Download the artifact from the repository provided above.
2. Install a recent version of OpenSSL.
3. Install Rust, version 1.75 or newer.
4. Install a recent version of Python 3 and the `numpy`, `pandas` packages (to parse benchmark results).

A.3.2 Basic Tests

1. Use `RUSTFLAGS='-C target-cpu=native' cargo test --lib` to run basic unit tests.
2. Use `RUSTFLAGS='-C target-cpu=native' cargo bench "CLMUL Multiplication"` to check that your machine offers hardware support for carry-less multiplication. If the test succeeds your machine has CLMUL support.

A.4 Evaluation workflow

To prepare for benchmarking proceed as follows:

1. Compile the benchmark binary using `RUSTFLAGS='-C target-cpu=native' cargo build -release --bin maestro --features="clmul"` (this assumes that the machine compiling and the benchmark servers are using the same architecture).
2. On each benchmark server setup the necessary TLS certificates.
3. If necessary, modify the network conditions.
   (a) Run `ip addr show` (or a similar command) to obtain the name of the network interface of each benchmark server, e.g., `eth0`, `enp0s3`, etc.
   (b) In case a previous experiment modified the network, remove the limitation before adding a new one: `tc qdisc del dev <iface name> root netem`
   (c) Set bandwidth and round trip time (RTT) as `tc qdisc add dev <iface name> root netem rate <bandwidth> delay <0.5 * RTT>`, for example `tc qdisc add dev eth0 root netem rate 200mbit delay 7.5ms` to limit the network traffic over eth0 to 200Mbit/s with a round trip time of 15ms.

4. Start the benchmark via CLI on all three server.

More details are given in the `README` file mentioned above.

A.4.1 Major Claims

(C1): *The artifact is functional. The provided source code compiles successfully and the unit tests run without errors. Using three terminals, one can run any of the implemented protocols on the same machine where communication is over localhost. This is proven by experiment (E1)*

(C2): *The throughput benchmarks are reproducible. That is one can replicate the relative throughput of the different protocols, i.e., claims of the form "Protocol X has twice the throughput than Protocol y". This is proven by experiment (E2).*

(C3): *The latency benchmarks are reproducible. That is one can replicate the relative latency of the different protocols. This is also proven by experiment (E2).*

A.4.2 Experiments

(E1): *[Run a small-scale benchmark: 20 human minutes + 10 compute minutes]* Experiment E1 builds upon the successful compilation of the source code and execution of unit tests (from Section A.3.2).
Set B to be the batch size, say 1000. Set R to be the number of repetitions, say 2.

1. Follow the description *Running benchmarks > On localhost* in the `README` file.
2. Open three terminals on the evaluation machine. Then run the benchmark binary with SIMD B and rep R, like this `target/release/maestro --config p1.toml --threads 4 --simd B --rep R --csv result-p1.csv chida lut16 gf4-circuit lut256 lut256-ss mal-chida mal-chida-rec-check mal-lut16-ohv mal-gf4-circuit-opt mal-lut256-ss-opt` (repeat for the second and third terminal with `p2.toml` and `p3.tomls`, and with `result-p2.csv` and `result-p3.csv`, respectively.
3. There should be information printed in all terminals about which protocol is currently benchmarked, etc.
4. When all finished successfully, it should say `Writing CSV-formatted benchmark results to result-p1.csv`.
5. Parse the benchmark data using `python parse-csv.py result-p1.csv result-p2.csv result-p3.csv`.

If all these steps ran without error and the parsing script shows throughput for the different protocols, this experiment is successful.

(E2): *[Run the Benchmarks] [2 human hours + 5 compute hour]: Experiment E2 is to run the benchmarking software given the different network conditions mention in Section A.2.5.*
Computing on larger batch sizes takes more time, in particular in networks with latency. Thus for (E2) we propose reduced parameter numbers and repetitions for each scenario.

- (Throughput) 10Gbit network: $B = 100000$ with $R = 5$.
- (Latency) 10Gbit network: $B = 1$ with $R = 10$.
- (Throughput) 1Gbit network: $B = 50000$ with $R = 5$.
- (Latency) 1Gbit network: $B = 1$ with $R = 10$.
- (Throughput) 200Mbit network: $B = 10000$ with $R = 5$.
- (Latency) 200Mbit network: $B = 1$ with $R = 10$.
- (Throughput) 100Mbit network: $B = 10000$ with $R = 5$.
- (Latency) 100Mbit network: $B = 1$ with $R = 10$.
- (Throughput) 50Mbit network: $B = 10000$ with $R = 5$.
- (Latency) 50Mbit network: $B = 1$ with $R = 10$.

Preparation: Setup the network and servers and prepare the benchmark software using the steps outlined in Section A.3.

Execution:

1. Setup the latency and throughput of the network.
2. Run the benchmark software on all three machines.
3. Repeat for each of the claimed network settings.

Results:

1. Parse the benchmark results using the given python script.
2. Compare the outcome with the original data in the `benchmark-data` folder. The relative differences in throughput and latency of the different protocols should roughly match.

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/usenixsec2025/.