USENIX Security '25 Artifact Appendix: "Shechi: A Secure Distributed Computation Compiler Based on Multiparty Homomorphic Encryption"

Haris Smajlović[*] University of Victoria
David Froelicher[*] MIT
Ariya Shajii Exaloop Inc.
Bonnie Berger MIT
Hyunghoon Cho Yale University
Ibrahim Numanagić University of Victoria

A Artifact Appendix

A.1 Abstract

Shechi is a Pythonic framework for high-performance secure distributed computing. Its artifact includes installation, setup, and performance comparison against the other frameworks for secure distributed computing. The comparison is done through four sets of experiments: (i) micro-benchmarks, (ii) basic workflows, (iii) complex workflows, (iv) scalability tests, and (v) ablation study. In the following sections, we provide detailed instructions on how to run each experiment.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no security nor ethical risks while executing this artifact.

A.2.2 How to access

Access Shechi's artifact at `https://zenodo.org/records/14868541`.

A.2.3 Hardware dependencies

Original Shechi experiments were evaluated in a distributed setup, each party on a different machine with 12-core Intel i7-8700 CPUs (3.20GHz), 64 GB RAM, all connected via a LAN network with 1 Gb/s bandwidth and 0.5 ms latency. For easier artifact evaluation, we have enabled all experiments to be run on a single machine, each party as a separate process communicating over UNIX sockets. The online run, however, is still possible.

Any machine with at least 64 GB RAM will suffice for most experiments. The only experiments that require more RAM (around 200 GB) are the scalability tests that involve more than 3 parties to run on a single machine (as a rough estimate, each party consumes around 20 GB of RAM for the heaviest experiments).

[*]Co-first authors.

A.2.4 Software dependencies

Shechi runs only on Linux at the moment (any maintained distribution). For reference, our experiments were run on CentOS Stream 9.

A.2.5 Benchmarks

We enabled all experiments to be executed on simulated datasets (generated at random in each run and simulating data similar to the lung cancer dataset from dbGaP (accession: phs000716.v1.p1)).

A.3 Set-up

A.3.1 Installation

We suggest following the installation instructions in the `README.md` in the artifact. The same instructions are provided here for completeness.

Install Codon first:
```
mkdir $HOME/.codon && curl \
-L https://github.com/exaloop/codon/releases/\
download/v0.17.0/codon-$(uname -s | awk \
'{print tolower($0)}')-$(uname -m).tar.gz | \
tar zxvf - -C $HOME/.codon --strip-components=1
```
And then install Sequre/Shechi:
```
curl \
-L https://github.com/0xTCG/sequre/releases/\
download/v0.0.20-alpha/sequre-$(uname -s | \
awk '{print tolower($0)}')-$(uname \
-m).tar.gz | tar zxvf - -C \
$HOME/.codon/lib/codon/plugins
```
Finally, add an alias for the sequre command (make sure it is a one-liner in the terminal):
```
alias sequre="find . -name 'sock.*' -exec rm {}
\; && CODON_DEBUG=lt $HOME/.codon/bin/codon run
--disable-opt="core-pythonic-list-addition-opt"
-plugin sequre"
```

A.3.2 Troubleshooting

Some Shechi experiments require Python. If Shechi fails to automatically link this dependency, make sure to export path to `libpython.so` shared library to `CODON_PYTHON` environment variable.

A.3.3 Basic Test

Run `sequre examples/local_run.codon` within the artifact. This will run the simple multiplication example in a simulated network of 3 parties. If the output contains the `MHE initialize` log for each party (CP), then Shechi has been successfully installed.

A.4 Evaluation workflow

Please note that all the commands below are also available in `USENIX25_README.md` in the root directory of the artifact.

A.4.1 Major Claims

(C1): The performance of Shechi's low-level operations is on par with the state-of-the-art frameworks and libraries. This is proven by the experiment (E1) described in Section 10.2.1, the results of which are reported in Table 2.

(C2): Shechi is up to 80× faster than the state-of-the-art homomorphic encryption frameworks when computing basic workflows like L2 distance and matrix multiplication. This is proven by the experiment (E2) described in Section 10.2.2, which is illustrated in Figure 7 (left).

(C3): Shechi is up to 15× faster than the state-of-the-art frameworks when computing complex, large-scale workflows such as Kinship estimation based on Genotypes, Principal Components Analysis, and Genome-Wisde Association Study. This is proven by the experiment (E3) described in Section 10.2.3, which is illustrated in Figure 7 (right).

(C4): Shechi's performance scales linearly, while the performance of Secure Multiparty Computation-based framework Sequre scales quadratically when increasing the number of computing parties. This is proven by the experiment (E4) described in Section 10.3, which is illustrated in Figure 8.

(C5): Shechi's ablation study shows that Aggregation and Encoding optimization is essential for good performance. This is proven by the experiment (E5) described in Section 10.4, which is illustrated in Figure 9 (right).

A.4.2 Experiments

(E1): [Micro-benchmarks] [5 human-minutes + 10 compute-minutes + 5GB RAM]: experiment evaluates the runtime of low-level operations in Shechi, Sequre, SEAL, MP-SPDZ, and Lattigo and demonstrates that Shechi's low-level functionalities are on par with the other frameworks.

Preparation: Make sure to run Shechi and Sequre experiments from the root directory of the artifact.

Execution: Shechi and Sequre: `sequre -release scripts/invoke.codon run-benchmarks --local --jit --lattiseq --mpc --mhe`. SEAL: `docker run --rm --privileged hsmile/seal:bench`. MP-SPDZ: `docker run --rm --privileged hsmile/mpspdz:bench`. Lattigo: `docker run -it --rm --privileged hsmile/lattigo:micro`.

Results: Inspect the `stdout` for reported runtimes for each benchmark and benchmarked procedure. The results should reflect the results reported in Table 2.

(E2): [Basic workflows] [5 human-minutes + 1.5 compute-hour + 10GB RAM]: experiment evaluates the runtime and network consumption of computing the L2 distance and matrix multiplication in Shechi, Sequre, HEFactory, EVA, MP-SPDZ, and Lattigo. It demonstrates that Shechi is constantly faster than other HE-based frameworks and consumes moderate bandwidth, as presented in Figure 7 (left).

Preparation: Make sure to run Shechi and Sequre experiments from the root directory of the artifact.

Execution: Shechi and Sequre: `sequre -release scripts/invoke.codon run-benchmarks --local --jit --stdlib-builtin`. HEFactory: `docker run --rm --privileged hsmile/hefactory:latest`. EVA: `docker run --rm --privileged hsmile/eva:bench`. MP-SPDZ: `docker run --rm --privileged hsmile/mpspdz:bench`. Lattigo: `docker run -it --rm --privileged hsmile/lattigo:bench`.

Results: Inspect the `stdout` for reported runtimes and network statistics for each benchmark and benchmarked procedure. Please note that HEFactory and EVA will not provide any network statistics. This is because they are executed in a centralized (non-distributed) fashion, under the assumption that all parties will encrypt and share their data beforehand. The results should reflect the results illustrated in Figure 7 (left).

(E3): [Complex workflows] [5 human-minutes + 18 compute-hours + 60GB RAM]: experiment evaluates the runtime and network consumption of computing the Kinship estimation based on Genotype, Principal Components Analysis, and Genome-Wisde Association Study in Shechi, Sequre, and Lattigo. It demonstrates that Shechi is constantly faster than other frameworks and consumes less bandwidth, as presented in Figure 7 (right).

Preparation: Make sure to run Shechi and Sequre experiments from the root directory of the artifact.

Execution: Shechi and Sequre: `sequre -release scripts/invoke.codon run-benchmarks --local --jit --king --pca --gwas-without-norm`. Lattigo Kinship: `docker run -it --rm --privileged` `hsmile/lattigo:king`. Lattigo GWAS (with PCA): `docker run -it --rm --privileged hsmile/lattigo:gwas`.

Results: Inspect the `stdout` for each benchmark's reported runtimes and network statistics at the end of the benchmarked procedure. Inspect the `results` directory to check the accuracy, or run `python scripts/accuracy.py` in the root directory of the artifact. Please note that the accuracy of Sequre KING may be off target. This is due to the SMC overflow that can happen for some randomly generated input. The results should reflect the results illustrated in Figure 7 (right).

(E4): [Scalability] [5 human-minutes + 8 compute-hours + 200GB RAM]: experiment evaluates the runtime and network consumption of computing the Principal Components Analysis in three different network setups: 2, 4, and 8 computing parties. It demonstrates that Shechi scales linearly as opposed to Sequre's quadratic performance drop when increasing the number of computing parties, as presented in Figure 8.

Preparation: Navigate to the root directory of the artifact.

Execution: 2-party setup: `SEQURE_CP_COUNT=3 sequre -release scripts/invoke.codon run-benchmarks --local --jit --pca`. 4-party setup: `SEQURE_CP_COUNT=5 sequre -release scripts/invoke.codon run-benchmarks --local --jit --pca`. 8-party setup: `SEQURE_CP_COUNT=9 sequre -release scripts/invoke.codon run-benchmarks --local --jit --pca`.

Results: Inspect the `stdout` for reported runtimes and network statistics. The results should reflect the results illustrated in Figure 8.

(E5): [Ablations] [1 human-minute + 40 compute-minutes + 5GB disk]: experiment evaluates the small ablation study for all possible optimization setups (with each of the three main optimizations either turned on or off). It demonstrates that the Aggregation and Encoding optimization (`enc` in `stdout`) is detrimental to performance and more effective than Maximizing efficient plaintext operations (`pri` in `stdout`) and Minimizing multiplication redundancy (`fac` in `stdout`), as illustrated in Figure 9 (right).

Preparation: Navigate to the root directory of the artifact.

Execution: Run `sequre -release scripts/invoke.codon run-benchmarks --local --jit --ablation`.

Results: Inspect the `stdout` for reported runtimes and network statistics. The results should reflect the results illustrated in Figure 9 (right).

A.5 Notes on Reusability

Shechi is a programming framework with Pythonic domain-specific language and, as such, is not restricted to specific experiments. It is intended for building arbitrary secure distributed applications, and we encourage the users to inspect the examples in `applications` and `stdlib/sequre/stdlib` directories for more examples of Shechi. We also plan to publish a comprehensive documentation and tutorial for Shechi in the near future.

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/usenixsec2025/`.