TVA: A multi-party computation system for secure and expressive time series analytics

Muhammad Faisal, *Boston University;* Jerry Zhang, *University of California San Diego;* John Liagouris, Vasiliki Kalavri, and Mayank Varia, *Boston University*

https://www.usenix.org/conference/usenixsecurity23/presentation/faisal

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artifact Appendix: TVA: A multi-party computation system for secure and expressive time series analytics

Muhammad Faisal
Boston University

Jerry Zhang[*]
University of California San Diego

John Liagouris
Boston University

Vasiliki Kalavri
Boston University

Mayank Varia
Boston University

A Artifact Appendix

A.1 Abstract

TVA is a multi-party computation (MPC) system for secure analytics on secret-shared time series data. Our work introduces new secure time series protocols that compute tumbling and session window operators. We implement these operators in the semi-honest setting using the *3PC replicated secret-sharing* protocol and in the malicious setting using the *4PC Fantastic Four protocol*. We run experiments to evaluate the performance of the newly introduced protocols using queries from real-world applications. The experiments require the deployment of two MPI clusters on AWS: (i) using machines in the same region (*LAN setting*) and (ii) using machines in different regions (*WAN setting*).

A.2 Description & Requirements

TVA is a typical MPC system that consists of *input parties*, *computing parties*, *output parties*. Our experiments are designed to evaluate the computation performed by the machines that host the computing parties. Before starting any experiment, the software needs to be deployed and initialized on each machine hosting a computing party to form a MPC cluster.

A.2.1 Security, privacy, and ethical concerns

There are no such concerns for our artifact deployment and evaluation. We emphasize that TVA is an academic proof-of-concept prototype and has not received careful code review. This implementation is NOT ready for production use.

A.2.2 How to access

We host our artifact on Github.

[*]Work completed at Boston University.

A.2.3 Hardware dependencies

TVA does not require special hardware and can operate on general-purpose CPU-based machines. For the experiments, we use two types of machines which are available on AWS: EC2 `r5.8xlarge` instances (32 vCPUS and 256GB RAM) and EC2 `r5n.16xlarge` instances (64 vCPUs and 512GB RAM). We use the `r5.8xlarge` machines when evaluating end-to-end latency for queries in the LAN and WAN settings (experiments E1-E4 in Section A.4.2). We use the `r5n.16xlarge` machines in the experiments that compare TVA with Waldo (experiments E5, E6 in Section A.4.2).

We provide SSH access to our AWS clusters so that the reviewers can reproduce the results using the same hardware and settings we used.

A.2.4 Software dependencies

Our artifact is implemented in `C++14`. The artifact requires two dependency packages: `libsodium` (1.0.18) and `MPICH` (3.3.2). We used Linux Ubuntu (20.04.4), which also requires installation of `CMake` (>=3.15.0) and `pkg-config` (>=0.29.2). The installation may require other packages based on the operating system version's pre-installed packages. For more details, check our dependency installation script `./scripts/setup.sh`.

A.2.5 Benchmarks

Our main performance results are based on two set of experiments. The first set evaluates the end-to-end latency of the real-world queries described in Section 6.2 in the paper. The results are shown in Figure 4. The second set of experiments is used to compare TVA's performance with the Waldo time series database. We report the comparison results in Table 2 and Figure 3.

TVA's workload consists of both local computation and communication among the parties. Depending on the network bandwidth and latency characteristics of the cluster, we categorize the experimental setting as follows:

- **Same Machine**: In this scenario, computing parties are deployed on the same machine. This is useful for testing the framework setup and protocol correctness. The experiments can be run with the following command.

```
cd build && cmake .. && make cloud
mpirun -np 3 ./cloud
```

- **LAN**: In this scenario, parties are deployed on different machines but in the same data center. For these experiments, we use the AWS `us-east-2` region. Network latency in this setting is less than `1ms`. The experiments can be run using the following command, where `machine-i` represents the machine of the corresponding computing party.

```
mpirun -np 3 -hosts \
machine-1,machine-2,machine-3 ./cloud
```

- **WAN**: In this scenario, we deploy computing parties on machines across different regions. Specifically, we use the following four regions: `us-east-2` (Ohio), `us-east-1` (Virginia), `us-west-1` (California), and `us-west-2` (Oregon). Network latency between machines in these experiments should be around `40ms`.

- **WAN-Simulated**: For simulating the same conditions as those Waldo has been evaluated in, we use a simulated WAN network where latency between computing parties is fixed at `20ms`. To achieve this configuration, deploy a LAN setting as described above and run the script *"./scripts/waldo_wan.sh"* to simulate the WAN.

A.3 Set-up

A.3.1 Installation

There are two phases to correctly set up TVA. Since the system consists of multiple computing parties (either 3 or 4 parties), we first need to install the framework on each party independently and then set up the MPI cluster.

Computing Party setup: In this step, we install the TVA source code and its dependencies. First, install the following dependencies:

- **building tools**: `cmake` and `pkgconfig`.
- **libsodium**: We use it for random number generation.
- **MPICH / OpenMPI**: We use it for establishing communication among the computing parties.

For instructions on how to install these dependencies on a linux distribution, see the script "`./scripts/setup.sh`".

After completing the above step, you should be able to use the system in the "Same Machine" setting, i.e., where all computing parties run in the same machine. At this point, you can run the tests to make sure everything is correct.

```
cd ./scripts
./run_tests.sh
```

Cluster setup: To use the framework in the LAN or WAN settings, we need to replicate the previous steps for each machine in the cluster. Once we have TVA working on every machine, we can start building the cluster as follows:

1. Make sure that machines have pair wise SSH access to each other. This step depends on the cloud service provider, as firewall settings and defaults are different.

2. Modify the `/etc/hosts` file on each machine to include other computing parties with names in the format `machine-i` for LAN, and `machine-wan-i` for WAN.

3. Build the code using either the semi-honest (**semi**) or the malicious protocol (**mal**) on each machine with the following script.

```
cd scripts
./build_experiments.sh semi  # mal
```

4. Run one of the experiments under `scripts` such as `cloud.sh` depending on which results you need to reproduce.

```
cd scripts
./cloud.sh semi lan
```

A.3.2 Basic Test

After finishing setting up a machine, test the framework setup using the following command:

```
cd scripts
./run_tests.sh
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): TVA's can successfully compute online queries with rigid time constraints and evaluate complex analytics on millions of input rows with modest use of resources. For this part, we need to reproduce the results in Figure 4 and Table 3.

(C2): For multi-predicate queries, TVA provides lower latency compared to Waldo both in the malicious setting and in the semi-honest setting. For this, we need to reproduce the results in Table 2.

(C3): For window queries, TVA is up to two orders of magnitude faster than Waldo, which becomes competitive only when the ratio of the window length over the whole time domain is relatively small. The results of this experiment are shown in Figure 3 and exact numbers are reported in Section 6.1.2.

A.4.2 Experiments

The following steps are required if using a private cluster. However, we can provide SSH access to our AWS machines and, in this case, you can start from the execution step directly.

For experiments E1 and E2, use the LAN setting described in Section A.2.5. For experiments E3 and E4, use the WAN setting. For experiments E5 and E6, use the WAN-Simulated. Use machines similar to machines `5.8xlarge` for experiments E1 through E4, and machines similar to machines `5n.16xlarge` for experiments E5 and E6.

(E1): *[Semi-honest LAN] [15 human-minutes + 1.5 compute-hour]*: In this experiment, we reproduce the results in Figure 4-a, which represents the latency for the 3 application queries in the semi-honest 3PC protocol when the cluster is deployed in the LAN setting.

Preparation: Prepare a cluster with three machines in the LAN setting as described in Sections A.2.5 and A.3.1.

Execution: Follow these steps:

1. On each machine, execute the following command to build the experiments files:

```
cd scripts
./build_experiments.sh semi
```

2. Use the corresponding bash file on just one of the 3 machines to start the experiment execution.

```
# On main machine of the cluster
cd scripts
./energy.sh semi lan
./cloud.sh semi lan
./medical.sh semi lan
```

Results: The results will be appended to the files in the results directory and you can compare the new results with the old ones at the beginning of the file.

(E2): *[Malicious LAN] [15 human-minutes + 3 compute-hour]*: Follow the same steps as in E1, except deploy 4 machines and replace the argument `semi` with `mal`.

(E3): *[Semi-honest WAN] [15 human-minutes + 5 compute-hour]*: Follow the same steps as in E1, except deploy the cluster in the WAN setting and replace the argument `lan` with `wan`. Make sure to update the hosts file using `machine-wan-i` as the host name.

(E4): *[Malicious WAN] [15 human-minutes + 10 compute-hour]*: Follow the same steps as in E2, except deploy the cluster in the WAN setting and replace the argument `lan` with `wan`. Make sure to update the hosts file using `machine-wan-i` as host the name.

(E5): *[Semi-honest Waldo] [15 human-minutes + 1 compute-hour]*: In this experiment, we reproduce the results in Figure 3 and Table 2.

Preparation: Prepare a cluster with 3 machines in the LAN setting as described in Sections A.2.5 and A.3.1. Use the script specified in WAN-Simulated to configure the network latency.

Execution: Follow these steps:

1. On each machine, execute the following command to build the experiments files:

```
cd scripts
./waldo_wan.sh S
./build_experiments.sh semi
```

2. Use the corresponding bash file on just one of the 3 machines to start the experiment execution.

```
# On main machine of the cluster
cd scripts
./waldo_energy_query.sh semi wan
./waldo_table_equality.sh semi wan
./waldo_table_greater.sh semi wan
```

Results: The results will be appended to the files in the results directory and you can compare the new results with the old ones at the beginning of the file.

(E6): *[Malicious Waldo] [15 human-minutes + 2 compute-hour]*: Follow the same steps as in E5 except use 4 machines and replace the argument `semi` with `mal`.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.