USENIX Security '24 Artifact Appendix: Scalable Multi-Party Computation Protocols for Machine Learning in the Honest-Majority Setting

Fengrun Liu Xiang Xie Yu Yu

A Artifact Appendix

A.1 Abstract

In this paper, we present a novel and scalable multi-party computation (MPC) protocol tailored for privacy-preserving machine learning (PPML) with semi-honest security in the honest-majority setting. Our protocol utilizes the Damgård-Nielsen (Crypto'07) protocol with Mersenne prime fields. By leveraging the special properties of Mersenne primes, we are able to design highly efficient protocols for securely computing operations such as truncation and comparison. Additionally, we extend the two-layer multiplication protocol in ATLAS (Crypto'21) to further reduce the round complexity of operations commonly used in neural networks.

In the artifact evaluation, we conduct performance evaluations for oblivious inference in various settings involving different number of parties, including 3-party computation(3PC), 7PC, 11PC, 21PC, 31PC, and 63PC. Our protocol is very scalable in terms of the number of parties involved. For instance, our protocol completes the online oblivious inference of a 4-layer convolutional neural network with 63 parties in 0.1 seconds and 4.6 seconds in the LAN and WAN settings, respectively. To the best of our knowledge, this is the first fully implemented protocol in the field of PPML that can successfully run with such a large number of parties. Notably, even in the three-party case, the online phase of our protocol is more than 1.4× faster than the Falcon (PETS'21) protocol.

A.2 Description & Requirements

We implement the protocol framework using approximately 14.2k lines of code (LOC) in C++. Our implementation leverages the communication backend of MP-SPDZ and the neural network frontend of Falcon. We conducted a performance evaluation for inference in various settings involving different numbers of parties, including 3-party(3PC), 7PC, 11PC, 21PC, 31PC, and 63PC to show the scalability of our framework.

A.2.1 Security, privacy, and ethical concerns

N/A

A.2.2 How to access

The artifact is open-sourced in Git repository https://github.com/f7ed/hmmpc-public with the stable URL https://github.com/f7ed/hmmpc-public/tree/b7d65e9d43bc3eb1610fc0000e895b8664df8b66 submitted in the Artifact Evaluation.

A.2.3 Hardware dependencies

We strongly recommend readers compile and run the program on Linux machines since we do not fully test the configurations on other platforms. We developed and tested this artifact in macOS with Intel processors but we did not test with Apple Silicon. The hardware-accelerated AES-NI instructions are required for efficient random number generation.

A.2.4 Software dependencies

As for Linux distributions, we recommend Ubuntu 20.04 LTS or 22.04 LTS as a platform, on which we conducted all the experiments and tests. We developed it on macOS Ventura 13.5 with Intel processors for reference. The artifact can be compiled by `g++`. For ease of implementation, we embed the code of the communication backend in MP-SPDZ and the neural network frontend in Falcon. Hence, it depends on OpenSSL for secure channels as in MP-SPDZ. It also depends on the Sodium library to generate randomness. To accelerate matrix multiplication, we utilize the Eigen library, the algorithms of which can use multi-threading with OpenMP. We enable OpenMP by default in our compilation. We have added Eigen's header files in our artifact, which are the only required files to compile with Eigen.

A.2.5 Benchmarks

We employ the widely used MNIST dataset, which consists of a collection of 28×28 pixel images depicting handwritten numbers. The objective is to accurately predict the corresponding number for each image. We select three standard neural networks from the field of privacy-preserving machine learning. Network-A is a 3-layer DNN network derived from SecureML. Network-B is a 3-layer CNN network derived from Chameleon. Network-C is a 4-layer CNN network derived from MiniONN. For more specific information and details about these neural networks, please refer to Appendix E of the paper.

A.3 Set-up

Follow the "Requirement" of README in the Git repository, which will guide you through setting up the required workspace. On Linux, you can execute a single command to install all the requirements: `sudo apt install -y build-essential cmake libssl-dev libsodium-dev iproute2`.

A.3.1 Installation

You can use the `make all` command to compile the program. The "Makefile" should automate most of the process. For ease of running multi-party computation, we support many `Bash` scripts to support local execution in a single machine and remote execution over multiple servers. The detailed instructions to execute locally and remotely are described in Section 3 and Section 4 of README, respectively.

We strongly recommend readers read Section 3.5, which describes the key commands executed in the `Bash` scripts and enable readers to fully understand what happens in the `Bash` scripts.

A.3.2 Basic Test

In order to check all the required dependencies are installed and the functioning is fine, readers can use `make -j8 network` and `make -j8 eigen` to test the communication module and the math module. Moreover, we provide a `Bash` script to test the primary protocols mentioned in the paper, enabling readers to catch how these protocols work and modify the source code easily. The unit tests for protocols are described in Section A.4.1.

A.4 Evaluation workflow

A.4.1 Major Claims

We support unit tests for the primary protocols proposed in the paper via a `Bash` script. You can modify the option to easily test the functionality of each protocol. The usage is `./test_unit.sh [test_name]`.

Thm 3.4: `./test_unit.sh Trunc` conduct a test for Protocol 3.1 in the paper. It tests the multiplication between secret fixed-point numbers and public fixed-point numbers, which requires performing a pure truncation.

Thm 3.5 `./test_unit.sh Fixed-Mult` conduct a test for Protocol 3.2 in the paper. It tests the multiplication between secret fixed-point numbers.

Thm 4.1 `./test_unit.sh PreOR` conduct a test for Protocol 4.1 in the paper. It tests for computing the prefix-OR over shared bits.

Thm 4.2 `./test_unit.sh Bitwise-LT` conduct a test for Protocol 4.2 in the paper. It tests for computing the result of bitwise less-than, given two bitwise sharings.

Thm 5.1 `./test_unit.sh DReLU` conduct a test for Protocol 5.1 in the paper. It tests for computing the derivation of ReLU of secret numbers, i.e. the sign bit.

Thm 5.2 `./test_unit.sh ReLU` conduct a test for Protocol 5.2 in the paper. It tests for computing the ReLU of secret numbers.

Thm 5.3 `./test_unit.sh Maxpool` conduct a test for the Protocol 5.3 in the paper. It tests for computing the Maxpool and its derivation of a vector of sharings.

A.4.2 Experiments

For ease of conducting experiments, we provide `Bash` scripts to support local inference in a single machine and remote inference over multiple servers. All the raw data points of Table 2-5 figured in the paper are collected in the Excel Tables located in here.

(E1): Perform oblivious inference locally on a single machine. The detailed functionality, output, and supported modifications are described in Section 3.

Preparation: N/A

Execution: `./Scripts/inference.sh <npc>` can simulate any arbitrary parties of odd numbers on a single machine, to perform the oblivious inference.

Results: It prints the statistical data of P_0 to the terminal. The output consists of two parts, the offline part and the online phase. Except for the time, the communication size, and the number of rounds, it also counts the number of different random sharings required in the online phase.

(E2): Perform oblivious inference remotely on multiple servers. The detailed steps and supported modifications are described in Section 4. Results of Tables 2, 3, 4 are conducted in this way.

Preparation: Follow Step 1-Step 3 in Section 4 to compile `inference.x` and use `tc` command to set latency and bandwidth for LAN or WAN. We offer various scripts to facilitate separate execution on multiple servers. All scripts are located in `Scripts/infer-<npc>` for $n = 3, 7, 11, 21, 31, 63$. Readers only need to modify this script to conduct the experiment with different settings.

Execution: Detailed instructions for the 7PC setting is illustrated in Step 4 - Step 5 in Section 4.

Results: It prints the simplified statistical data of P_0 to the specified location, including the time and the number of communication bytes.

A.5 Notes on Reusability

We recommend readers understand the meanings of variables/fields in the scripts, which greatly facilitate you to conduct experiments with desirable settings. More details about these modifiable fields are described in Section 3.3 and Step 5 in Section 4.2 of README.

Note that all the raw data points in the Excel tables (located in here) are collected in the setting of TRUE_OFFLINE=1 where the random sharings are generated in a separate preprocessing phase totally before the online phase, and the communication size is the average of the communication bytes sent by all parties. Hence, if you want to reproduce the result, you need to set TRUE_OFFLINE=1.

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/usenixsec2024/`.