ACSAC 2024 Artifact Documentation: Privacy-Preserving Verifiable Neural Network Inference Service

Arman Riasi
Virginia Tech
armanriasi@vt.edu

Jorge Guajardo
Robert Bosch LLC — RTC
jorge.guajardomerchan@us.bosch.com

Thang Hoang
Virginia Tech
thanghoang@vt.edu

I. Abstract

Our vPIN prototype is a privacy-preserving and verifiable CNN inference service designed for Machine Learning as a Service (MLaaS) settings. It ensures the privacy of a client's data during CNN inference while simultaneously guaranteeing the verifiability of the inference results. The artifact submission includes the complete source code of vPIN , written in Python and Rust, totaling approximately 5,200 lines of code. The source code is intended to be executed on a Linux-based operating system, specifically tested on a server machine with a 48-core Intel CPU and 256 GB of RAM. The primary dependencies include Python 3.8 or later, Rust 1.72.0-nightly, and the necessary cryptographic libraries. All software dependencies and installation instructions are documented in the `README.md` available at https://github.com/vt-asaplab/vPIN/blob/main/README.md.

II. Description & Requirements

A. Security, privacy, and ethical concerns

Our artifact uses standard cryptographic libraries and widely-used Python packages, all of which can be easily installed using `pip`. The execution of this artifact does not pose any risks to the evaluators' system security, data privacy, or ethical concerns.

B. How to access

The source code for our implementation can be found at https://github.com/vt-asaplab/vPIN/tree/ACSAC_2024.

C. Hardware dependencies

A server machine with a 48-core Intel CPU (Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz) and 256 GB of RAM.

D. Software dependencies

We used the following dependencies in our implementation:

- **Python 3.8+**
- **Rust (rustc 1.72.0-nightly)**
- **Cargo**: Used for Rust package management.
- **Required Python Packages**:
  - `torch`: 2.4.0
  - `torchvision`: 0.19.0
  - `torchmetrics`: 1.4.1
  - `ecdsa`: 0.19.0
  - `numpy`: 2.0.1

Specifically, we used the `python-ecdsa` library to generate cryptographic keys based on elliptic curves (EC). For client-server communication, we used the standard Python `socket` library. We used the Rust `libspartan` library to convert our arithmetic constraints into a rank-1 constraint system (R1CS) and to implement our underlying CP-SNARK scheme. Specifically, employed the SpartanDL variant, which internally leverages the Hyrax polynomial commitment and the `curve25519-dalek` library for EC operations.

E. Modifications to the Spartan Library

This project uses the Spartan repository, located in the `src/proof_generation/spartan/` directory, as a core library for generating proofs for our witnesses. To integrate Spartan with the vPIN framework, we made proof-related functions public to enable direct calls from the vPIN framework. Additionally, we incorporated the `commitment_test.rs` file, located in `src/proof_generation/vPIN_proof_generation/src/`, extracted from the ezDPS repository, for generating commitments to auxiliary witnesses.

III. Set-up

A. Installation

We have provided a detailed `README.md` file, available at https://github.com/vt-asaplab/vPIN/blob/main/README.md, which contains all necessary instructions to set up and run our source code. In summary, running `pip3 install -r requirements.txt` will automatically install all the Python packages with the specified versions needed to run the vPIN code. To generate proofs, it is necessary to install `Rust` and `Cargo`. The instructions for installing Rust and Cargo are provided in the `README.md` file at https://github.com/vt-asaplab/vPIN/tree/main?tab=readme-ov-file#installing-rust-and-cargo.

B. Port Configuration

Our experiments use a range of default ports:

- 35000–35006: for individual tests
- 36000–36004: for batch CNN experiments
- 37000–37011: for batch convolution experiments

These default port numbers can be customized by modifying lines 29–39 in the `script.sh` file.

C. Output Configuration

To control the verbosity of the output during the execution of experiments, the `script.sh` includes a `QUIET_MODE` setting:

- `QUIET_MODE=1` (default): Outputs are logged to files, and terminal output is suppressed to minimize clutter.
- `QUIET_MODE=0`: Enables verbose output, displaying detailed information directly in the terminal for real-time monitoring.

To switch to verbose mode, set `QUIET_MODE=0` in the `script.sh` file at line 46.

IV. Evaluation workflow

Ensure that `pip3 install -r requirements.txt` has been successfully executed before proceeding.

- **Experiment E1:**
  - **Resource Requirements and Duration:** This experiment requires 3 GB of RAM and approximately 50 minutes to complete. Make sure you have enough disc space, as the generated table will be 230 MB in size.
  - **Execution Instructions:** Execute the following command:
    ```
    $ ./script.sh -b
    ```
  - **Results:** This script generates a table using the baby step giant step algorithm for computing the EC discrete logarithm used in the decryption phase on the client side.
- **Experiment E2:**
  - **Resource Requirements and Duration:** This experiment requires approximately 3 GB of RAM and takes between 6 to 20 minutes to complete, depending on the network type (A, B, ..., E). Using the `-t` option to run all CNN networks sequentially requires 16 GB of RAM and takes 75 minutes to complete.
  - **Prerequisite:** Before proceeding with Experiment E2, ensure that Experiment E1 has been executed to precompute the necessary table required for computing the EC discrete logarithm in the client-side code.
  - **Execution Instructions:** Execute the following command, adjusting the parameter value after `-c` to specify the network type. Available options are {-A, -B, -C, -D, -E}. For example:
    ```
    $ ./script.sh -c -A
    ```
    or
    ```
    $ ./script.sh -c -D
    ```
  - **Batch Testing:** To run all CNN networks sequentially, use the `-t` option:
    ```
    $ ./script.sh -c -t
    ```
  - **Results:** This experiment generates all witnesses in the `vPIN/src/rust_files` directory. Then, it runs the Rust code to generate proofs and displays the proving time, verification time, and proof size, as illustrated in Figure 2.
- **Experiment E3:**
  - **Resource Requirements and Duration:** This experiment requires 2 GB of RAM and takes 2 to 60 minutes to complete, depending on the input image size and filter size. Using the `-t` option to run all convolution experiments sequentially requires 5 GB of RAM and takes about 4 hours to complete.
  - **Prerequisite:** Before proceeding with Experiment E3, ensure that Experiment E1 has been executed to precompute the necessary table required for computing the EC discrete logarithm in the client-side code.
  - **Execution Instructions:** Execute the following command, adjusting the parameter value after `-d` to specify the convolution filter size. Available options are {3, 5, 7}, and then specify the input image size from $\{32 \times 32, 64 \times 64, 128 \times 128, 256 \times 256\}$. For example:
    ```
    $ ./script.sh -d 3 32
    ```
    or
    ```
    $ ./script.sh -d 7 128
    ```
  - **Batch Testing:** To run all convolution experiments sequentially for all combinations of filter sizes (3, 5, 7) and image sizes (32, 64, 128, 256), use the `-t` option:
    ```
    $ ./script.sh -d -t
    ```
  - **Results:** This experiment generates all witnesses in the `vPIN/src/rust_files` directory. Then, it runs the Rust code to generate proofs and outputs the proving time, verification time, and proof size, as shown in Figure 3.
- **Experiment E4:**
  - **Resource Requirements and Duration:** This experiment requires 230 GB of RAM and approximately 4 hours minutes to complete.
  - **Prerequisite:** Before proceeding with Experiment E4, ensure that Experiment E1 has been executed to precompute the necessary table required for computing the EC discrete logarithm in the client-side code.
  - **Execution Instructions:** Execute the following command:
    ```
    $ ./script.sh -l
    ```
  - **Results:** This experiment generates all witnesses in the `vPIN/src/rust_files` directory for the LeNet model inference scheme. Then, it runs the Rust code to generate proofs and outputs the proving time, verification time, and proof size for each layer, as reported in Table 2.
- **Experiment E5:**
  - **Resource Requirements and Duration:** This experiment requires a few MB of RAM and approximately 50 minutes to complete.
  - **Execution Instructions:** Execute the following command:
    ```
    $ ./script.sh -a
    ```
  - **Results:** This script evaluates the impact of using a fixed-point representation on the inference accuracy of the LeNet model on the MNIST dataset, as detailed in the Accuracy section of Section 6.3.

V. Notes on Reusability

Our source code for vPIN is a proof-of-concept prototype and is not ready for production use.