Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning

Hidde Lycklama, *ETH Zurich;* Alexander Viand, *Intel Labs;* Nicolas Küchler, *ETH Zurich;* Christian Knabenhans, *EPFL;* Anwar Hithnawi, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity24/presentation/lycklama

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: "Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning"

Hidde Lycklama[1], Alexander Viand[2], Nicolas Küchler[1], Christian Knabenhans[3], Anwar Hithnawi[1]

[1]*ETH Zurich* [2]*Intel Labs* [3]*EPFL*

A Artifact Appendix

This repository contains the artifact for the USENIX 2024 submission "Holding Secrets Accountable: Auditing Privacy-Preserving Machine Learning".

A.1 Abstract

In this work, we introduce Arc, an MPC framework designed for auditing privacy-preserving machine learning. Arc cryptographically ties together MPC training, inference, and auditing phases to allow robust and private auditing. At the core of our framework is a new protocol for efficiently verifying inputs against succinct commitments, ensuring the integrity of the training data, model, and prediction samples across phases. We evaluate the performance of our framework when instantiated with our consistency protocol and compare it to hashing-based and homomorphic commitment-based approaches, demonstrating that it is up to $10^4\times$ faster and up to $10^6\times$ more concise.

A.2 Description & Requirements

The artifact is a prototype implementation of the Arc framework. The prototype is designed to focus on evaluating the overheads of input consistency protocols for MPC computations. The framework uses the MPC implementation of MP-SPDZ, a research framework execute MPC programs with different protocols. Arc extends the MPC protocols in MP-SPDZ with the ability to check that MPC inputs are consistent with a commitment through a novel consistency check protocol.

A.2.1 Arc Components

Our implementation consists of the following components.

- MPC auditing functions Implementations of several auditing functions (in addition to ML training and inference) in MP-SPDZ's DSL.
- MPC consistency utils A helper library with 1) functionality to load the correct datasets and models for auditing and 2) to compute the correct metadata for the inputs and outputs to run the consistency check protocol on.
- Consistency check protocol The code for the consistency check protocol based on polynomial commitments. This component uses Arkworks' poly_commit library and is implemented in Rust.
- Share conversion and efficient EC-MPC scripts Our framework adds functionality to MP-SPDZ for share conversion and efficient EC-MPC operations, which are implemented as lower-level MP-SPDZ scripts.

A.2.2 Arc Benchmarks & Evaluation

- DoE-Suite The evaluation is built using the DoE-Suite, which allows straightforward reproducibility of results by defining experiments in a configuration file (*suite*) and executing them on a set of machines. We provide the necessary DoE-Suite commands to reproduce all results. However, it is also possible to obtain the individual commands used to invoke the framework and run them manually.
- Utils Experiment runner utility that ties together the MPC computation, the share conversion and the consistency protocols. This script is the entrypoint for the remote servers when running the experiment and reads the experiment instance config file created by doe-suite for each experiment.

A.2.3 Security, privacy, and ethical concerns

There are no concerns when running this artifact locally. Please note that executing experiments on your AWS infrastructure involves the creation of multiple EC2 instances, resulting in associated costs. Please manually check that any created machines are terminated afterward.

A.2.4 How to access

The artifact can be accessed at `https://github.com/pps-lab/arc/tree/ae_final`

A.2.5 Hardware dependencies

None.

A.2.6 Software dependencies

No private software is required for this artifact. This artifact has been tested on Ubuntu and MacOS. The artifact relies on DoE-Suite to install all necessary dependencies on the end-to-end servers. To run DoE-Suite, we require Python 3.9, Poetry, AWS CLI and Make to be installed, which are also highlighted in the installation instructions below. The framework uses Poetry to manage further Python dependencies. The sub-components require additional dependencies, which must be installed manually if you wish to run these components locally (without DoE-Suite). In particular, mpc-consistency requires Rust to be installed.

A.2.7 Benchmarks

No proprietary benchmarks and public datasets are automatically loaded by the build scripts.

A.3 Set-up

We provide a `make` command to run a JupyterLab notebook (artifact.ipynb) to run the experiments and evaluate the artifact. *We strongly recommend following the detailed instructions in this notebook that is part of the artifact.* This ensures that the environment is set up correctly and that the necessary dependencies are installed.

A.3.1 Installation

We require Python, Poetry and Make to be installed to run the artifact. To get a local copy up and running follow these steps.

1. Local clone of the repository (**with submodules!**)
   ```
   git clone --recurse-submodules
   git@github.com:pps-lab/arc.git
   ```
2. Install Python Poetry
   ```
   curl -sSL https://install.python-poetry.org
   | python3 -
   ```
3. Install Make
4. Install Install AWS CLI version 2 (to run remote experiments on AWS)

Environment Variables The doe-suite requires a few environment variables and should handle the rest of the configuration automatically (including for the Jupyter notebook) using relative paths and poetry. Setup environment variables for the Doe-Suite are displayed in the jupyter notebook.

```
# Root project directory (expects
# the doe-suite-config dir in this folder)
export DOES_PROJECT_DIR=

# Your unique short name, such as your
# organization's acronym or your initials.
export DOES_PROJECT_ID_SUFFIX=ae
```

For AWS EC2:

```
export DOES_CLOUD=aws

# Name of ssh key used for setting up access
# to aws machines (name of key not path)
export DOES_SSH_KEY_NAME=id_ec_arc
```

`DOES_SSH_KEY_NAME` refers to the key the reviewers have received through artifact evaluation system.

> *Tip: To make it easier to manage project-specific environment variables, we recommend a tool like Direnv. Direnv allows to create project-specific .envrc files that set environment variables for specific working directories. With Direnv, the below environment variables would be set in doe-suite/.envrc*

Setting up AWS Authentication details can be found in the Artifact submission system. This will allow the Artifact reviewers to run the evaluation on the same resources stated in the paper submission. The experiments on AWS are automated with DoE-Suite and can be called from the JupyterLab environment. Reviewers should have received a private key: `id_ec_arc` and AWS credentials

1. Move the provided private key `id_ec_arc` to the `.ssh` folder of your home directory (reviewers should have received the key, otherwise contact us). Ensure the id_ec_arc key has the correct permissions: `shell    chmod 600 ~/.ssh/id_ec_arc`
2. Configure AWS credentials using `aws configure`. The credentials can be found in the `arc_ae_accessKeys.txt`. Set `eu-central-1` as the default region. By default, credentials should be stored in `~/.aws/credentials`.
3. To configure SSH access to AWS EC2 instances, you need to add a section to your `~/.ssh/config` file:
   ```
   Host ec2*
      IdentityFile ~/.ssh/id_ec_arc
      User ubuntu
      ForwardAgent yes
      StrictHostKeyChecking no
      UserKnownHostsFile=/dev/null
   ```

For more details on the installation of doe-suite please refer to the doe-suite documentation and AWS-specific instructions.

Running JupyterLab From this point on it is possible to run the Jupyter notebook which contains the experiments and evaluation of the artifact. To launch the Jupyter notebook in the environment with the correct dependencies, we provide a make command.

1. Navigate to the `doe-suite` sub-directory and run `make jupyter` which will launch these instructions in the form of a notebook.
2. Run the first code cell in the notebook to check that it prints *Environment loaded successfully*.

If you see any errors, make sure the correct environment variables are set.

A.3.2 Basic Test (AWS, 30 minutes)

To test that your local machine is configured properly and that you have access to the AWS resources, you can run the following command (with `doe-suite` as working directory):

```
make run suite=audit_fairness id=new
```

which will launch two sets of servers on AWS to reproduce the fairness experiments (Fig. 6, column 1). The command will run the experiments, fetch the results and store them in the `doe-suite-results` folder.

A.3.3 Basic Test (Local, 15 minutes)

We also provide a `Makefile` in the project's root directory to run scripts locally. For this, we require MP-SPDZ dependencies to be installed. We refer to the MP-SPDZ documentation for more details on those dependencies. To install the framework's dependencies in the project directory, run `make install`.

Datasets You must store the relevant datasets in the `MP-SPDZ/Player-Data` directory. We provide the datasets from the paper, preprocessed to work with MP-SPDZ, in a public s3 bucket at `http://pps-mpspdz-data.s3.amazonaws.com/{DATASET_NAME}.zip`. Available datasets are: `adult_3p`, `mnist_full_3party`, `cifar_alexnet_3party`. For the QNLI dataset, the identifier is `glue_qnli_bert` but the data will be loaded by the compilation script so there is no need to download it. Then run one of the tasks with the following command:

```
poetry run make ring script=inference \
dataset=adult_3p
```

The framework will compile the script, compile the MP-SPDZ binaries (which can take tens of minutes) and then run the script.

A.4 Evaluation workflow

We assume that the following steps are followed within the JupyterLab environment (artifact.ipynb). This artifact relies on the DoE-Suite to manage experiment configurations, orchestrate the execution of experiments, collect the results and post-process them into tables and plots.

The doe-suite can be run using a set of commands defined in a Makefile in the `doe-suite` directory that invoke Ansible. Use `make run` to run experiments, from now on referred to as *suites*, that are defined in the doe-suite-config/designs directory. Each suite defines a set of Ansible roles that are used to configure the remote machines running Ubuntu 22.04. Results of these experiments are then combined together into plots that are defined in the doe-suite-config/super_etl directory.

For each result shown in the paper, we have a separate section that contains:

1. Code to create and display the plot shown in the paper and corresponding dataframe based on the output files from the benchmarks (stored in `doe-suite-results-cameraready`). These files can be downloaded from this polybox.
2. The command to reproduce the results on AWS. You can uncomment the command and run the cell with Ctrl + Enter. Due to the large amount of output and long running time, we recommend to run these commands in a separate terminal window. The results from these experiments will be stored in `doe-suite-results` and will appear in a separate set of plots in the notebook.

A.4.1 Major Claims

This work introduces two major claims:

(C1): Arc instantiated with our consistency protocol is up to 10^4x faster and 10^6x more concise than hashing-based (SHA3) and homomorphic commitment-based (PED) approaches.

(C2): Across all settings, Arc significantly outperforms related approaches in terms of runtime, with a storage overhead comparable to the hash-based approach.

Both claims are proven by the experiments in Section 6: **Training** (E1, Fig. 4), **Inference** (E2, Fig. 5) and **Auditing** (E3, Fig. 6).

A.4.2 Experiments

For each of training, inference and auditing, we provide a table of suites that belong to this setting in the Jupyter notebook. Each row contains a rough estimate of the maximum duration of running that suite. This estimate is based on the raw runtimes in the benchmark logs, but the estimation of the overhead of creating and provisioning the machines may not be completely accurate. To run a suite, simply select a suite from the table and invoke doe-suite to run it. We provide an option to run the suite inline, or in a separate terminal window.

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/usenixsec2024/`.