USENIX'23 Artifact Appendix: A Verified Confidential Computing as a Service Framework for Privacy Preservation

Hongbo Chen[1], Haobin Hiroki Chen[1], Mingshen Sun[2], Kang Li[3], Zhaofeng Chen[3], and XiaoFeng Wang[1]

[1]*Indiana University Bloomington, {hc50,haobchen,xw7}@iu.edu*
[2]*Independent Researcher, bob@mssun.me*
[3]*CertiK, {kang.li, zhaofeng.chen}@certik.com*

A Artifact Appendix

A.1 Abstract

We propose a security protection principle for confidential computing, Proof of Being Forgotten (PoBF). It has two requirements: NoLeakage and NoResidue. These properties are formalized and proven under an abstract model for Trusted Execution Environment (TEE) in Coq. On the other hand, we implement a prototype PoBF-Compliant Framework (PoCF), which provides a framework to conduct Confidential Computing as a Service (CCaaS). These prototypes come with a verifier that can prove some properties specified in PoBF are satisfied. Besides, PoCF can support various real-world applications and the protections introduced in PoCF incur minor runtime performance overhead.

A.2 Description & Requirements

Reproducing the exact experiment results needs special hardware support: processors with Intel SGX and AMD SEV instruction extension support.

A.2.1 Security, privacy, and ethical concerns

Our artifacts come with no security, privacy, or ethical concerns. However, since building it requires plenty of dependencies and runtimes, we suggest reproducing the experiments in a non-production environment. We also provide access to an experiment virtual and/or physical machine.

A.2.2 How to access

Our code is published on GitHub and can be accessed via the link: `https://github.com/ya0guang/PoBF/tree/usenix-sec-ae`.

A.2.3 Hardware dependencies

The SGX-related experiments require an Intel processor with SGX instruction extension, and SEV-related experiments require an AMD processor with SEV instruction extension. Running the multi-threading test requires at least 32GB RAM (EPC Size) and we recommend using servers with at least 64GB RAM.

A.2.4 Software dependencies

PoCF requires dependencies from the system software and many dependencies for different platforms, so we recommend following the build instructions in our `README.md` in the GitHub repository or **just running the script located under the root directory, named `setup.sh`**. We only list the general software dependencies here.

Common Dependencies.

- Linux OS, preferably `Ubuntu 20/22.04 LTS`
- Rust `nightly-2022-11-01`
- `python3`: $\geq$ v3.8, $<$ v3.11
- `jupyter` notebook
- `mirai` abstract interpreter (version recorded in the script)
- `prusti` verifier (version recorded in the script)
- `tvm` v0.12.0
- `llvm` $\geq$ v10.0
- `Coq` proof assistant $>$ v8.13

SEV Specific Dependencies.

- Azure's SEV Guest attestation library. See `https://github.com/Azure/confidential-computing-cvm-guest-attestation`.

SGX Dependencies.

- SGX Driver, not needed if kernel > 5.11
- SGX SDK v2.17.101
- Teaclave Rust SGX SDK and Intel SGX SDK for Linux
- Attestation dependencies, including `aesm`, `dcap` or `epid`
- (*Optional*) other TEE frameworks: `enarx`, `gramine`, `occlum`

A.2.5 Benchmarks

Our benchmark data or the build scripts are included in the repository. We develop several confidential computing tasks that may require specific data and/or models. Workloads are all in `cctask/` folder, and the corresponding data (generators) are `data/`.

- `tvm` task requires `resnet152` and a picture input.
- `db` requires a `ycsb` client for generating, loading, and querying the database. This client is a submodule in our repository.
- `fasta` and `fann` requires generated sequence and an input number (serialized as a little-endian byte array) respectively.
- Other tasks require a dummy data payload.

A.3 Set-up

A.3.1 Installation

Please follow the `README.md` at https://github.com/ya0guang/PoBF/tree/usenix-sec-ae to set up the software environment. If you use the (virtual) machine provided by us, you can ignore this step.

A.3.2 Basic Test

- To make sure SGX functions correctly, check `sudo service aesmd status` and confirm it is successfully serving. Also, you should see SGX-related devices when using `ls /dev | grep sgx`.
- If TVM is installed and configured properly, one should be able to compile the TVM library under `cctasks/evaluation_tvm/model_deploy`. It can be checked by simply executing `make -j`.
- Rust programs should work if compilation does not fail.
- Coq works if `coqc` command can be executed.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The Coq proof is machine-checked.
(C2): PoCF verifier works on the sample project.
(C3): The protections in the PoCF introduce minor runtime performance overhead compared to NATIVE setting.
(C4): The tasks are indeed supported by PoCF.

A.4.2 Experiments

Note: For the comprehensive instruction, please check `scripts/README.md` in our repository.

E1 and E2 are verification of PoCF, and we expect the experiment to be passing the verification. E3 and E4 are performance evaluations that can be performed on Intel SGX platform and/or AMD SEV platform. We have scripts for both single- and multi-threaded experiments. We expect that the protection introduced by POCF is minor compared to NATIVE.

(E1): [Coq Proof Checking] [3 minutes]: successfully compilation implies successful proof verification.
How to: Just compile the Coq source code.
Execution: Run `coqc *.v` at `pobf_proof/`
Results: Successful compilation.

(E2): [PoCF Verification] [5 human-minutes + 30-60 compute-minutes]: Verify the implementation of PoCF. The verifier invokes `mirai` and `prusti` to conduct NOLEAKAGE and NORESIDUE checkings.
How to: Please follow the steps in `Verification towards PoCF` in our `README.md`.
Preparation: Install `mirai` and `prusti` using the scripts. This takes some time to compile from the source code.
Results: There are two cases for a negative case and a positive case: one contains threats and one does not. One can try to remove the `verified_log!` in the source file `src/userfunc.rs` to see the difference

(E3): [Overhead Analysis (microbenchmarks)] [15 human-minutes + 1 compute-hour + 1GB disk]: Confirm the overhead introduced by PoCF protections is minor.
How to: First, compile and run the microbenchmarks. Then analyze the data and generate the figure. This evaluation is performed in single- and multi-threading scenarios.
Preparation: Compile the microbenchmark tasks.
Execution: Run the evaluation scripts that can be found under `scripts/evaluation.sh`. Remember to set the `task` variable to the task `polybench`. It will perform 10 repetitions (or more if you need). The cost breakup results would be automatically printed to the console if you are evaluating the PoBF task. For the stack page number microbenchmark, please refer to `README.md`.
Results: First execute the code in the Python notebook

under `scripts/figure.ipynb`. The script draws the figures that show the performance of PoCF and Native on different tasks. We expect the performance (execution time) of PoCF to be better. For other microbenhmarks, please copy-and-paste the results to the Jupyter notebook and visualize them.

(E4): [Real-world application (macrobenchmarks)] [30 human-minutes + 1 compute-hour + 5GB disk]: Confirm the overhead introduced by PoCF protections is minor.
How to: First compile and run the confidential computing tasks. Then analyze the data and generate the figure. This evaluation is also performed in single- and multi-threading scenarios.
Preparation: Compile the macrobenchmark tasks.
Execution: For the KVDB task: Please execute the YCSB client that is included as a submodule in the repository. Follow the instruction in the `README.md`. You may need to first load the data by the workload `workload/load.toml` and then execute the corresponding workload A and C. For other tasks: Please execute the evaluation script `scripts/evaluation.sh`.
Results: Execute code in the Python notebook `scripts/db.ipynb`, `scripts/figure.ipynb` The script draws the figures that show the performance of PoCF and Native on different tasks. We expect the performance (execution time) of PoCF to be better. For the results collected from YCSB, please copy-and-paste the results to the plotting script.

A.5 Notes on Reusability

- If you want to modify the state transitions, edit `pobf_state/src/task.rs`.
- If you want to add/modify the CC Task, follow the examples in `cctasks`. You may also want to modify the build script.
- You could also change the verification options by modifying `pobf_verifier/pobf-verify`

Note that our artifact is an academic project. Any use of the code should adhere to the license.

A.6 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/.