A Hardware-Software Co-design for Efficient Intra-Enclave Isolation

Jinyu Gu, Bojun Zhu, Mingyu Li, Wentai Li, Yubin Xia, and Haibo Chen, *Shanghai Jiao Tong University*

https://www.usenix.org/conference/usenixsecurity22/presentation/gu-jinyu

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

This document demonstrates the artifact evaluation of LIGHTENCLAVE, which uses MPK to provide intra-enclave isolation within SGX enclaves. We incorporate LIGHTENCLAVE into two SGX libOSes (Graphene-SGX and Occlum) and carry out evaluations to show the performance of Graphene-SGX/Occlum with and without LIGHTENCLAVE. We provide a remote machine as the SGX, PKU (MPK), and MPX CPU features are needed. According to the specification, AE reviewers can firstly build the tested applications and libOSes, and then carry out all experiments mentioned in paper. The experiments will reproduce the results and generate figures and tables in the paper.

We do not apply for the Available badge because one of the founders currently does not allow to open source the work.

A.2 Artifact check-list (meta-information)

- **Program:** The libOSes used in the experiment are Occlum (commit 0a06c898) and Graphene-SGX (commit 9c226c9a). The applications and libraries used in the experiments are SGX-OpenSSL (commit 5bacfaf), SGX-SQLite3 engine (v3.23.0), Lighttpd (v1.4.40), GCC (v4.4.5), Fish shell (v3.0.0) and BusyBox (v.1.23.1)
- **Hardware:** An Intel x86 platform that supports Intel SGX, PKU and MPX.
- **Run-time environment:** The experiment is carried out on Linux. The kernel should set *CR4.FSGSBASE = 1* to allow userspace applications use *wrfsbase* and *wrgsbase* to modify fs.base and gs.base. On newer versions of Linux (>= 5.9), *CR4.FSGSBASE = 1* is always set. The SGX SDK 2.4 and SGX Driver 2.4 should be installed on the host. Docker is used as the building environment. We also use the runtime environment provided by Occlum and Graphene-SGX.
- **Metrics:** We use the applications' throughput and execution latency of operations to study LIGHTENCLAVE's performance.
- **Output:** Some of the outputs are numerical results that can be compared with tables in the paper. The other outputs are figures that are available in the paper.
- **Experiments:** We provide scripts that reproduce the experiment results and generate figures and tables in the paper.
- **How much disk space required (approximately):** Around 18G. We suggest different AE reviewers use different working directories. Since the disk space of our remote machine is limited, please remove the working directory once the artifact evaluation completes (in case leading to out-of-disk for others).
- **How much time is needed to prepare workflow (approximately):** We provide a remote machine which is setted up for artifact evaluation so that reviewers do not need to prepare the workflow.
- **How much time is needed to complete experiments (approximately):** The building procedure is about 1 hour. The complete evaluation takes about 3 hours.

A.3 Description

A.3.1 How to access

N/A

A.3.2 Hardware dependencies

The artifact evaluation requires an Intel x86 platform that supports Intel SGX, PKU and MPX. Our remote machine has an Intel i7-10700 IceLake CPU.

A.3.3 Software dependencies

Except for the environment mentioned in the checklist, LIGHTENCLAVE requires the building system and toolchain from Occlum and Graphene-SGX. To save the time for reviewers, we have prepared the software dependencies for the artifact evaluation in the offered machine.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

The installation procedure requires building the libOSes (Graphene and Occlum with and without LIGHTENCLAVE) and the compilation toolchain. After that, we compile the applications for the evaluation. We provide a remote machine with software dependencies prepared, where reviewers can start from the installation stage. Please check *Artifact access* in the submission (on hotcrp) for detail.

After login into the machine, the home directory contains the *lightenclave-artifact* directory that holds evaluation materials. Before starting the evaluation, please copy *lightenclave-artifact* to another directory to avoid conflicts between different reviewers.

```
> sudo cp -r lightenclave-artifact your-
    evaluation-directory
> cd your-evaluation-directory
```

We firstly build Occlum toolchain and applications running inside Occlum. We use docker as the building environment.

```
> bash build_occlum_apps_in_docker.sh
# Now inside the docker
# Takes long time: about 45 minutes
> bash occlum/build_toolchain_and_app.sh
> exit
```

Then we build applications running inside Graphene using another docker environment. This docker environment is used in the

following building and evaluation workflow, including building Occlum and Graphene-SGX libraries and applications running without libOSes (for Figure 8).

```
> bash reproduce_in_docker.sh
# Now inside the docker
> bash graphene-sgx/build_app.sh
> bash occlum/build_libos.sh
> bash graphene-sgx/libos/build_libos.sh
> bash sdk-bench/prepare.sh
```

A.5 Experiment workflow

N/A

A.6 Evaluation and expected results

We claim that: 1. LIGHTENCLAVE is fast in terms of light-enclave creation and communication. 2. LIGHTENCLAVE incurs low performance overhead for intra-enclave isolation to applications. 3. LIGHTENCLAVE improves the performance in real-world scenarios in existing LibOSes.

For the 1st claim, Table 2 shows the task creation latency in SGX libOSes. When incorporated with LIGHTENCLAVE, the application creation time is shortened. The results can be reproduced by executing:

```
# about 8 minutes
> ./scripts/table2.py
```

Then Figure 7 demonstrates that LIGHTENCLAVE can provide fast enclave communication using shared memory between light-enclaves. In contrast, the communication in Graphene is more time-consuming due to data encryption. The figure can be reproduced by executing:

```
# Takes about 50 minutes
> ./scripts/figure7.py
> gnuplot -p ./plots/figure7.plt
# The figures locate at plots/figure7a.eps
    and plots/figure7b.eps
```

For the 2nd claim, we use LIGHTENCLAVE to isolate sensitive code from third-party code for security. We compare it with Nested Enclave, which uses an inner enclave to isolate third-party code. Figure 8a isolates OpenSSL library from the application. Figure 8b isolates SQLite3 library from a key-value store server. The figure can be reproduced by executing:

```
# Takes about 7 minutes
> ./scripts/figure8a.py
# Takes about 3 minutes
> ./scripts/figure8b.py
> gnuplot -p ./plots/figure8a.plt
# The figure locates at plots/figure8a.eps
> gnuplot -p ./plots/figure8b.plt
# The figure locates at plots/figure8b.eps
```

For the 3rd claim, we apply LIGHTENCLAVE to Occlum and Graphene and test real-world applications' performance. The applications are Lighttpd, GCC, Fish Shell and some serverless functions.

We configure Lighttpd with two isolated workers and use ApacheBench to get the throughput. LIGHTENCLAVE improves performance in Occlum since there is no boundary checking. Figure 9 shows the results, which can be reproduced by executing:

```
# Takes about 4 minutes
> ./scripts/figure9.py
> gnuplot -p ./plots/figure9.plt
# The figure locates at plots/figure9.eps
```

The fast task creation in LIGHTENCLAVE benefits GCC, which frequently forks processes for compilation. We isolate each GCC-related processes (*cc1*, *as*, *collect2* and *ld*) in the enclave. And we compile five applications with various sizes of code bases. Figure 10 shows the results, which can be reproduced by executing:

```
# Takes about 16 minutes
> ./scripts/figure10.py
> gnuplot -p ./plots/figure10.plt
# The figures locate at plots/figure10a.eps
    and plots/figure10b.eps
```

We then evaluate Fish Shell's performance by invoking several BusyBox commands (od, sort, grep, wc etc.) for text processing. LIGHTENCLAVE improve the performance by creating tasks fast (compared with Graphene) and avoiding SFI overhead (Compared with Occlum). Table 3 shows the result, which is reproduced by:

```
# Takes about 8 minutes
> ./scripts/table3.py
```

The fast task creation in LIGHTENCLAVE reduces initialization overhead in FaaS scenarios. We evaluate four serverless functions' execution latency to show the benefits. LIGHTENCLAVE is compared with initializing a new enclave before execution (COLD) and using an existing enclave for execution (WARM). In theory, LIGHTENCLAVE and WARM have similar execution latency while it takes fewer resources. Figure 11 shows the results, which can be reproduced by:

```
# Takes about 40 minutes
> ./scripts/figure11.py
> gnuplot -p ./plots/figure11.plt
# The figure locates at plots/figure11.eps
```

A.7 Experiment customization

N/A

A.8 Notes

If the experiments freeze (the execution time is far beyond the time we offer), reviewers can kill the docker and restart the experiment. Maybe the libOSes with the specified commits contain some unknown issues.

```
# Press Ctrl+C. Or use docker kill command
> exit
# Re-enter the docker
> bash reproduce_in_docker.sh
# e.g., ./scripts/figure10.py fails
```

```
> ./scripts/figure10.py
```

After the artifact evaluation, please remove the working directory as it consumes large disk space.

```
> sudo rm -rf your-evaluation-directory
```

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.