COMP4801 Final Year Project
Interim Report

Project Title: Analyzing and Improving the Performance of SGX
Department of Computer Science
The University of Hong Kong

Author: Fung Yuk Leung
UID: 3035277559

Supervisor
Dr. Cui H.M.
Doctor of Department of Computer Science

Creation Date: 15/1/2019
Last Revised: 15/1/2019
Version: 1.0
ABSTRACT

Software Guard Extensions (SGX) is an Intel product that provides strong security guaranty for confidential data. However, it results in performance degradation at the same time because it requires extra effort to run SGX. This project uses the method of control experiment, to analyze the performance of SGX in both fundamental operations (such as ecall and ocall) and actual applications (such as ZooKeeper, Memcached and Lighttpd) using the latest version of SGX Software Development Kit (SDK). At the current stage, this project investigated that there are totally 3 overheads that result in the performance degradation, which are switching, data transferring and memory access. They consumed 13473, 2537, 755 instruction cycles respectively. Although the analysis of performance in actual applications is not yet finished, the statistics obtained at this stage can help programmers decide whether it is worthy and comparable to apply SGX with declined performance in their product. Additionally, this project will suggest some improvements for SGX to enhance the performance at the next stage, for instance, remove the zeroing step in ocall with option out. These suggestions will be proved to be effective using the method of control experiment.
ACKNOWLEDGMENT

This research is supported/partially supported by Dr. Cui H.M and his assistant, Jian Yu. I thank them for providing insight and expertise to help me set up scientific experiments so that I can get concrete statistics to analyze the performance of Software Guard Extensions.
# TABLE OF CONTENTS

## INTRODUCTION ............................................................................................................. 8

- Background .................................................................................................................. 8
- Major Concern and Current Research .......................................................................... 9
- Scope and Contributions of The Project ...................................................................... 10

## OBJECTIVES ............................................................................................................... 11

## LITERATURE REVIEW .............................................................................................. 12

## METHODOLOGY ....................................................................................................... 13

- Main method ............................................................................................................... 13
- Experiment environment .............................................................................................. 13

## LIMITATIONS AND DIFFICULTIES ...................................................................... 14

## RESULTS AND FINDINGS .......................................................................................... 15

- Analysis of performance in fundamental operations ..................................................... 15
- Analysis of performance in actual applications ........................................................... 21
- Improvement ................................................................................................................. 21

## FUTURE PLANNING .................................................................................................. 22

## CONCLUSION ............................................................................................................. 23

## REFERENCES ............................................................................................................. 24
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SGX creates a trusted zone, called <em>enclave</em>, to isolate confidential data from untrusted environment, including operating system, virtual machine manager and firmware.</td>
<td>8</td>
</tr>
<tr>
<td>2. The manufacturer of SGX works as a trusted intermediary agent between users and service providers.</td>
<td>9</td>
</tr>
<tr>
<td>3. The program of experiment group in analyzing the overhead of switching to SGX mode.</td>
<td>15</td>
</tr>
<tr>
<td>4. The program of control group in analyzing the overhead of switching to SGX mode.</td>
<td>15</td>
</tr>
<tr>
<td>5. The experiment in analyzing the overhead of <em>ecall</em> with option <em>in</em>.</td>
<td>17</td>
</tr>
<tr>
<td>6. The mathematic model of <em>ecall</em> with option <em>in</em>.</td>
<td>18</td>
</tr>
<tr>
<td>7. The mathematic model of <em>ecall</em> with option <em>out</em>.</td>
<td>18</td>
</tr>
<tr>
<td>8. The mathematic model of <em>ecall</em> with option <em>in&amp;out</em>.</td>
<td>18</td>
</tr>
<tr>
<td>9. The mathematic model of <em>ocall</em> with option <em>in</em>.</td>
<td>18</td>
</tr>
<tr>
<td>10. The mathematic model of <em>ocall</em> with option <em>out</em>.</td>
<td>18</td>
</tr>
<tr>
<td>11. The mathematic model of <em>ocall</em> with option <em>in&amp;out</em>.</td>
<td>18</td>
</tr>
<tr>
<td>12. The mathematic model of consecutive read.</td>
<td>20</td>
</tr>
<tr>
<td>13. The mathematic model of non-consecutive read.</td>
<td>20</td>
</tr>
<tr>
<td>14. The mathematic model of consecutive write.</td>
<td>20</td>
</tr>
<tr>
<td>15. The mathematic model of non-consecutive write.</td>
<td>20</td>
</tr>
</tbody>
</table>
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Slopes of the mathematic models of <code>ecall</code> with option <code>in</code>, <code>ecall</code> with option <code>out</code>, <code>ecall</code> with option <code>in&amp;out</code>, <code>ocall</code> with option <code>in</code>, <code>ocall</code> with option <code>out</code>, and <code>ocall</code> with option <code>in&amp;out</code>.</td>
<td></td>
</tr>
<tr>
<td>2. Slopes of the mathematic models of consecutive read, non-consecutive read, consecutive write and non-consecutive write.</td>
<td>21</td>
</tr>
</tbody>
</table>
ABBREVIATIONS

SGX: Software Guard Extensions
SDK: Software Development Kit
INTRODUCTION

Background

Cloud computing (e.g. Google Cloud Platform) is a shared pool of computing resources. It is highly popularized and commercialized nowadays because of its advantage of lower cost of computation and storage. In cloud computing, users send their data to service providers, who process these data on behalf of users and then return corresponding results. A problem arises in such a model, which is the leakage of data due to irresponsible or untrusted service providers.

Software Guard Extensions (SGX), is invented to solve this problem. On the cloud server side, SGX creates a processor-hardened container, called enclave, to isolate confidential data inside it from untrusted environment (see Figure 1) [2]. Additionally, the manufacturer of SGX works as a trusted intermediary agent between users and service providers [6]. As Figure 2 shows, it measures the safety of execution environments of service providers using software attestation (a
technique for attesting authority), and then sends the corresponding results to users so that they can know whether they are communicating with a safe and trusted device.

Figure 2. The manufacturer of SGX works as a trusted intermediary agent between users and service providers.

Major Concern and Existing Studies

While providing strong security guarantee, SGX leads to a concern about performance. SGX is completely implemented with microcodes (the lowest specified level of processor and machine instructions sets) [6]. Running these microcodes requires extra efforts, which are instruction cycles (the basic operational process of a computer system). That is, SGX results in an certain overhead and hence lower performance than regular programs [5]. For example, while SGX create an enclave to protect confidential data, it divides a program into two parts. One part containing secret data and some important functions is placed in the enclave, another part containing regular functions is placed outside the enclave. However, functions in a same program interact with each other often and hence they need to cross the border of enclave, which results in degradation of performance.

The trade off between security level and performance is an extremely important consideration in software development. Therefore, some concrete statistics about the performance of SGX should be provided to software developers so that they can decide whether to apply SGX in their
products, based on these statistic as well as their software development requirements. Although there are some existing studies analyzing performance of SGX, they are contradictory with each other. For example, [3] investigated that programs using SGX ran 55% slower in some actual applications while [4] claimed that they ran just 12% slower. This gap implies that more researches are needed to prove which one is accurate. Lastly, existing studies (such as [3] and [5]) measured the performance of SGX with old versions of Software Development Kit (SDK), but Intel launched a latest version, 2.3.101, on October 18th, 2018. With new version of SDK, the performance of SGX might improve. Thus, there is a demand for researches on latest version.

**Scope and Contributions of The Project**

This project focuses on the performance of SGX, especially in applications of ZooKeeper, Memcached and Lighttpd. ZooKeeper is an open-source for maintaining configuration information and providing distributed synchronization and group services. It is selected by this project because it is widely used nowadays. Memcached is a key-value database and it is chosen since it is commonly used in caching between website servers. Lighttpd is a light-weight web server. It is selected because it is popularized and serve many concurrent requests. In conclusion, performance of these three applications are analyzed because they are representative of performance of cloud services.

This project makes two contributions. First, it investigated concrete statistics describing the performance of SGX (partially finished at current stage). Specifically, it provided the number of percentages describing how programs applying SGX cost more instruction cycles than regular ones, which helps programmers decide whether to apply SGX in their products. Comparing to existing studies, these statistics provide latest analysis on latest version of SGX SDK. Also, they help solve the contradiction between current results as mentioned before. Second, this project will give some suggestions of improvement to reduce the overhead and hence heighten the performance of SGX (to be finished at the next stage).
OBJECTIVES

The first objective of this project is to investigate the performance of SGX with latest version of SDK, especially in three applications, which are ZooKeeper, Memcached and Lighttpd.

The second objective of this project is to suggest any feasible improvement on SGX, especially on its interface, so that the performance of SGX can be heightened.
LITERATURE REVIEW

If old versions of SGX SDK are considered, there are already some existing studies analyzing their performance. For example, [4] investigated that programs applying SGX ran 12% slower in some actual applications; [3] suggested that the performance degradation might be as high as 79% in applications with high system call frequency, such as Memcached and OpenVPN; [5] indicated that the perform of SecureKeeper (an enhanced version of ZooKeeper applying SGX) was 11% slower than ZooKeeper (an open-source that provides centralized infrastructure and services).
METHODOLOGY

Main method
This project will use the method of control experiment. For each separate experiment, one group of programs, the experimental group, will apply SGX with SDK version 2.3.101 while another one, the control group, will just use regular techniques. In this project, each program will be run for 10,000 times to reduce errors or residuals. For each execution of a program, the real time stamp counter (RDTSCP, an instruction to obtain the current cycle) will be used to measure consumed instruction cycles because this is the most precise way to measure the running time. The performance of SGX will be reflected by the difference of results between the experimental group and control group.

Since each program will be run for 10,000 times, there will be 10,000 results. In this situation, their median is chosen to represent the final result. Median instead of average is selected because median can get rid of extreme values but average is influenced by extremely small or extremely large values.

Experiment environment
The above programs will be run in an environment with Intel Core i7-6700k 4GHz with 4 hyper-threaded cores, disabled dynamic frequency and voltage scaling, and an operating system of Ubuntu 14.04 LTS. I7-6700k is selected because only 6th (or later) generation of Intel Cores support SGX [2].
LIMITATIONS AND DIFFICULTIES

Since the experiment environment in this project has smaller size of RAM and fewer CPUs than actual server, the results of this project might have a certain level of residual, however, it can still give a reasonable estimation of true value.

When executing programs in experiments, there is a certain possibility that context switch to the operating system (an unpredictable event triggered by accidents) happens and hence contaminates the measurement. If this kind of event happens during experiments, the running time is longer than actual one. For solution, the negative influence can be minimized by running each program for 10,000 times. This is because context switch to the operating system happens rarely, so the majority of 10,000 executions are not affected and hence the negative influence is reduced and negligible.
RESULTS AND FINDINGS

Analysis of performance in fundamental operations
There are totally three kinds of fundamental operations that results in the performance degradation of SGX, which are switching to SGX mode, passing data into/extract data out of SGX enclave, and memory access in the SGX enclave. This project conducted different control experiments to analyze the running time of these overheads.

1. switching to/switching out of SGX mode
Switching to/switching out of SGX mode is triggered when dealing with confidential data inside the enclave (other modes have no right to process data inside the enclave). It results in an overhead because there are some checking of authorization before switching, which requires extra time to run.

![Figure 3. The program of experiment group in analyzing the overhead of switching to SGX mode.](image)

![Figure 4. The program of control group in analyzing the overhead of switching to SGX mode.](image)
Figure 3 shows the work flow of experiment group in analyzing the overhead of switching to SGX mode. First, the timer starts. Then the program calls a function that is defined inside an SGX enclave. At this moment, the computer switches to the SGX mode. The called function has no parameter, does nothing and returned immediately. Then the computer switches back to original mode. Finally, the timer ends. In this situation, the difference between the two timers, which is the result (27556 instruction cycles), is the total running time of the timer itself, a function call and two times of switching (switching into enclave and switching out of enclave). In theory, it is better if the timer ends inside the enclave instead of in the main function. In this case, the running time of switching is counted only once (only switching into the enclave). However, the timer is not allowed to run inside the enclave. Therefore, this project can only make the timer end in the main function and count the running time of switching twice (switching into enclave and switching out of enclave).

The result of experiment group includes not only the running time of two times of switching, but also the running time of the timer and a function, which is not in interest. Thus, a control group is required. Figure 3 shows the work flow of the control group. It is the same as the one of experiment group except that the function called in control group is not defined inside the enclave. Thus, there is no switching to SGX mode or switching out of SGX mode when it is called. In this situation, the result of control group (610 instruction cycles) is the running time of the timer and a function call. Then the result of experiment group subtracted by the result of control group is the running time of two times of switching (27556 – 610 = 26946 instruction cycles) and hence the running time of one switching is 13473 instruction cycles.

2. passing data into/extract data out of enclave

Passing data into/extract data out of SGX enclave is triggered by the data interactions between the enclave and the outside world. Data cannot pass the border of enclave directly. Instead, the transportation of data is done by creating a new copy of the data on another side of enclave. For example, when some data is passed from the outside world to the enclave, SGX will create a new file inside the enclave and then copy the data into it, which costs extra time.
There are two kinds of data interactions for SGX, which are `ecall` (from the outside world into the enclave) and `ocall` (the reverse direction of `ecall`, from the enclave to the outside world). For each of them, there are totally four options, which are `user_check`, `in`, `out`, and `in&out`. `User_check` is a special case as there is no copying involved and hence there is no overhead for both `ecall` with option `user_check` and `ocall` with option `user_check`. Therefore, this project did not analyze this option.

![Diagram](image)

**Figure 5.** The experiment in analyzing the overhead of `ecall` with option `in`.

Figure 5 shows the work flow of experiment in analyzing the overhead of `ecall` with option `in`. There are several different groups in this experiment. Different groups have different sizes of data in the data interaction. For example, the first group passes a 0-KB buffer of data into the enclave and the second group passes a 2-KB buffer of data into the enclave. Since the only difference between different groups is the size of data, the difference of results between different groups is caused by the difference of sizes of data. Thus, a mathematic model, whose independent variable is the size of data and dependent variable is the total running time, is obtained. With similar experiments, mathematic models of `ecall` with option `out`, `ecall` with option `in&out`, `ocall` with option `in`, `ocall` with option `out`, and `ocall` with option `in&out` are also obtained.
Figure 6 – Figure 11 shows the obtained mathematic models of *ecall* with option *in*, *ecall* with option *out*, *ecall* with option *in*out, *ocall* with option *in*, *ocall* with option *out*, and *ocall* with option *in*out respectively. All of them are linear, which show that the running time is in proportional to the size of data. The range of size of data in experiments of *ocall* is smaller than the one of *ecall*. This is because the buffer of *ocall* is allocated on the untrusted stack with
similar mechanism with ‘alloca’ but not ‘malloc’ [6]. As the size of stack is limited, the size of buffer of `ocall` and its range are limited as well.

<table>
<thead>
<tr>
<th>Name</th>
<th>Slope</th>
</tr>
</thead>
<tbody>
<tr>
<td>ecall with option in</td>
<td>1429.7</td>
</tr>
<tr>
<td>ecall with option out</td>
<td>1995.9</td>
</tr>
<tr>
<td>ecall with option in&amp;out</td>
<td>2308.9</td>
</tr>
<tr>
<td>ocall with option in</td>
<td>3054.1</td>
</tr>
<tr>
<td>ocall with option out</td>
<td>2249.7</td>
</tr>
<tr>
<td>ocall with option in&amp;out</td>
<td>4188.6</td>
</tr>
</tbody>
</table>

Table 1. Slopes of the mathematic models of `ecall with option in`, `ecall with option out`, `ecall with option in&out`, `ocall with option in`, `ocall with option out`, and `ocall with option in&out`.

Table 1 show the slopes of the above mathematic models that are obtained in this project. The slope represents the running time of passing every 1-KB of data with a specific option. Therefore, the overhead of passing data into/extract data out of SGX enclave is 2537 instruction cycles on average (for passing every 1-KB of data).

3. memory access in the SGX enclave

SGX maintains an integrity tree to ensure confidentiality, integrity and anti-roll-back protections [3], so accessing (read/write) memory inside the enclave needs to go through/modify the tree, which is a kind of overhead.
Figure 12 – Figure 15 show the mathematic models of consecutive read, non-consecutive read, consecutive write and non-consecutive write respectively. These models are obtained by measuring different running time of different size of target memory. All of them are linear, which show that the running time is in proportional to the size of target memory.

Table 2 shows the slopes of the above mathematic models that are obtained in this project. The slope represents the running time of accessing every 1-KB of data. Therefore, the overhead of passing data into/extract data out of SGX enclave is 755 instruction cycles on average (for passing every 1-KB of data).
4. conclusion for fundamental operations
In conclusion, from the aspect of fundamental operations, there are totally 3 overheads that result in the preform degradation, which are switching, data transferring and memory access. They consumed 13473, 2537, 755 instruction cycles respectively.

Analysis of performance in actual applications
The analysis of performance in actual applications is not yet done at current stage and this becomes a part of future planning.

Improvement
As for suggestions of improvement, this project suggested an intuitive idea to enhance the performance of SGX. When crossing the border of enclave, if ocall with option out is used, a buffer outside enclave memory will be first zeroed and then used to receive data from inner enclave. However, zeroing such a buffer cannot increase security guaranty. This is because even if the buffer is not zeroed first and its data get leaked, this will not damage the security since all confidential data are placed in enclaves and those leaked data are not secret. The performance of SGX will improve if this step is removed. This project will prove this suggestion to be effective using method of controlled experiment and suggest more advice at the next stage.

<table>
<thead>
<tr>
<th></th>
<th>consecutive</th>
<th>non-consecutive</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>505.8</td>
<td>615.3</td>
</tr>
<tr>
<td>write</td>
<td>867.3</td>
<td>1029.9</td>
</tr>
</tbody>
</table>

Table 2. Slopes of the mathematic models of consecutive read, non-consecutive read, consecutive write and non-consecutive write.
FUTURE PLANNING

I divide the next stage into three periods. Firstly, from 20/1/2019 to 20/2/2019, I will finish the analysis of performance in actual applications, which are ZooKeeper, Memcached and Lighttpd. I will conduct one or more separate control experiments for each application. In a control experiment, only the experiment group will apply SGX. Then, I will compare the running time of experiment group and the control group to analyze the degradation of performance.

Secondly, from 21/2/2019 to 1/4/2019, I will suggest more improvement on SGX to enhance its performance. I will prove my suggestions to be effective using the method of control experiment. In this case, the experiment group will become a group of programs applying my suggestions and the control group will become a group of programs applying original SGX. For expect result, the running time of experiment group should be shorter than the one of control group.

Finally, from 1/4/2019 to the end FYP period, I will the complete research paper and give a full presentation of it in the final presentation.
CONCLUSION
This project aims to analyze the performance of Software Guard Extensions (SGX) using control experiment, and to provide suggestions of improvement on SGX. At the current stage, this project investigated that there are totally 3 overheads that result in the preform degradation, which are switching, data transferring and memory access. They consumed 13473, 2537, 755 instruction cycles respectively. The analysis of performance in actual applications is not yet finished. This project also suggested an intuitive idea, removing the zeroing step in ocall with option out, to enhance the performance of SGX. The statistics obtained in this project can help programmers decide whether it is worthy and comparable to apply SGX with declined performance. Since the experiment environment has smaller size of RAM and fewer CPUs than actual server due to limitation of laboratory, which results in residuals to true value, future researches equipped with real server can be done to obtain the true value of degradation of performance.
REFERENCES


