On the Performance of Intel SGX

ChongChong Zhao[1], Daniyaer Saifuding[1], Hongliang Tian[2], Yong Zhang[2], ChunXiao Xing[2]
1.School of Computer and Communication Engineering, University of Science and Technology, Beijing 10083, China
2.Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
daniyar1815@gmail.com

***Abstract*—As cloud computing is widely used in various fields, more and more individuals and organizations are considering outsourcing data to the public cloud. However, the security of the cloud data has become the most prominent concern of many customers, especially those who possess a large volume of valuable and sensitive data. Although some technologies like Homomorphic Encryption were proposed to solve the problem of secure data, the result is still not satisfying. With the advent of Intel SGX processor, which aims to thoroughly eliminate the security concern of cloud environment in a hardware-assisted approach, it brings us a number of questions on its features and its practicability for the current cloud platform. To evaluate the potential impact of Intel SGX, we analyzed the current SGX programming mode and inferred some possible factors that may arise the overhead. To verify our performance hypothesis, we conducted a systematic study on SGX performance by a series of benchmark experiments. After analyzing the experiment result, we performed a workload characterization to help programmer better exploit the current availability of Intel SGX and identify feasible research directions.**

Keywords—Cloud Computing; Data Security; Cloud Security; Intel SGX; Performance Evaluation; Trusted Hardware

I. INTRODUCTION

Cloud computing has been credited with increasing competitiveness through cost reduction, greater flexibility, elasticity and optimal resource utilization. It provides users and enterprises with strong capabilities to store and process their data in third-party data centers. However, storing data or host application in the public cloud also mean losing control of the servers that may contain sensitive information. Although users of cloud computing infrastructure may expect their data to remain confidential, the fact is that today's clouds platforms are based on a classical hierarchical security model, which means it will just protect the privileged code from untrusted code [1]. As for the attacks that come from the privileged code like virtual machine monitors (VMM), there is still not a good solution to protect the user data. As a result, application developers must trust the users of client platforms and system administrators of cloud service platforms to follow security best practices that keep application-managed data safe from malicious software. However, the news like Google Engineer spied on teenager Gmail chat [2] and clerk of Switch Bank revealed customers' information [3] make us hold a suspect attitude towards the administer of the cloud platform. The survey also shows that related concern to the cloud security is a significant factor limiting cloud adoption [4]. Therefore, one of the biggest challenges for today's cloud environment is to provide a buildable solution to help cloud customers and enterprises ensure their security of data.

Although there were some technologies proposed to solve the problem of cloud security, the result is still not satisfying. The current best practice for protecting confidential data in the cloud is hardware security module (HSM) [5]. These dedicated appliances rely on tamper-proof hardware to protect critical secrets such as keys and support a range of cryptographic functions. However, HSM caused a large amount of extra cost and could not usually run general-purpose applications [1]. Previous research relied on trusted hypervisors to protect an application from a malicious system [6,7], but could not protect against a hypervisor controlled by a malicious or compromised cloud provider. Besides, Sumeet Bajaj and other people purposed using secure coprocessors (SCPU) to implement the confidential of data [8]. However, the secure co-processors are very expensive for lots of developers and have limited resource. In conclusion, today there is still no complete and cost-efficient solution to ensure the confidentiality of user data in the cloud environment. There is a clear need for technological solutions that help software developers or any customer of the cloud platform can safely manage their personal, financial or any sensitive data without encumbering their user experience, or limiting their person control.

In order to meet the needs above, the largest processor company Intel has developed Intel Software Guard Extensions (SGX) processor recently, a set of new instruction and memory access changes added to the Intel Architecture [9]. The SGX allows a process to instantiate a secure region of address space known as an enclave. An enclave is a protected area in the application address space, which provides confidentiality and integrity even in the presence of privileged malware [10]. Attempted accesses to the enclave memory area from software or system from the untrusted part are prevented even from privileged software such as virtual machine monitors, BIOS, or operating system function operating at a higher privilege level.

The advent of SGX raises a number of questions concerning the future of cloud security: What are limitations of current SGX programming mode? How the performance of an application that used SGX processor? How large are the cost to deploy SGX with current cloud platform? As for the fact that there is no any performance evaluation with the entity of SGX hardware before, we design and conduct a systematic study to find out the answers to the previous questions.

In this work, our main contribution is analyzing the current mode to use SGX in cloud system and summarizing its

978-1-5090-5437-4/16 $31.00 © 2016 IEEE
DOI 10.1109/WISA.2016.45

deficiency and technical difficulty. Besides, we adopt various kinds of ways to evaluate the practical performance of SGX and measure the actual cost by conducting the benchmark of SGX in the real application. Through analyzing different benchmark results, we also give more details about the practicability of using Intel SGX to solve the cloud security problem currently. Ultimately, we indicate some possible future research direction of using SGX to through solve the security of cloud environment.

II. SGX Programming Mode

The architecture of SGX protects the security of cloud data both against software attack and hardware attack. First, an enclave is created by an ECREATE instruction, which initializes a control structure in protected memory [10]. The enclave is protected by the processor via controlling access to enclave memory. Regardless of current privilege level and CPU mode (user-mode, kernel-mode, SMM, VMM or another enclave) [15], any instructions that attempt to read or write the memory of a running enclave from outside the enclave will fail. Meanwhile, the codes inside the enclave that attempt to access the system would also fail. Besides, to protect SGX against known hardware attacks, the enclave memory is encrypted using industry-standard encryption algorithms [12] with reply protection. Tapping the memory or connecting the DRAM modules to another system will only give access to the encrypted code. Therefore, the security of the data is assured by the SGX special mode for both software attacks and hardware attacks.

SGX is designed to protect the application by separating them in a trusted secure region of address space known as an enclave, however, full applications may have some properties that make them challenging to execute in the enclave. There are many instructions, especially some systemic instructions are not allowed to directly executed inside enclave [10]. While as for some most common workloads in the cloud platform, like a web server and data server, one of their apparent characteristics is system-intensive. They keep receiving users' data or posting them back to the users with the high frequency of I/O manipulation. In conclusion, restricting them from manipulating system instructions is obviously unpractical.

To implement porting existing application into SGX enclaves, the designer of Intel SGX proposed a new way of programming mode [11], which divides the application code into two logic components:

- Trusted Component: The code accesses the confidential data inside. This component is also called an enclave. More than one enclave can exist in an application.
- Untrusted Component: The rest of the application including all its modules.

From the cloud application perspective, making an enclave call (ECALL) appears as any other function call when using the untrusted part code. In the same time, the enclave code can leave the protected memory region and call the function in the untrusted zone by a special instruction, which called OCALL (see Figure 1). With this mode, users get a chance to successfully manipulate the system call or other necessary instructions that are not supported inside the enclave.

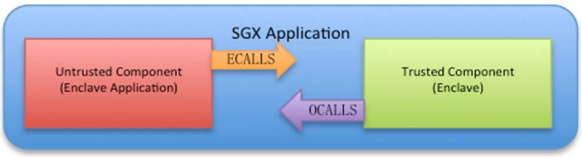

Fig. 1. Connection of the untrusted component and trusted component

From the SGX programming mode we discussed above, it is reasonable to suspect the SGX may arise some unnecessary overhead. Firstly, as for the reason that enclave memory keeps encrypted as it running, the ability of SGX to have memory access and dynamic memory allocation is highly questionable. Moreover, each crossing of enclave means the transition of processor mode. However, these transitions will definitely arise some unnecessary overhead by some process, like saving or restoring the register and flushing the CPU pipeline and even may pollute the microarchitecture data structure [15]. Therefore, the transitions into and out of enclave may be another crucial overhead for the performance on SGX.

III. Experiment Study

To verify the performance hypothesis that put forward above, we decide to measure the specific overhead of the SGX performance by designing and conducting systemic benchmark experiment. Our experiments run on the system comprised of a 4-core Intel Core i5-6200U SGX CPU running at 2.4GHZ with 8GB od 1600 MHz DDR3 RAM. The system type is 64-bit Windows Server 2016 Technical Preview 5.

A. Performace Cost: overhead comparison between calls

As we mentioned before, crossing from an enclave must need the transition of processor status, which may cause a certain amount of overhead. To measure the specific overhead of crossing enclaves, our first experiment measures the cycles per operation with different four kinds of calls: function call, system call and ECALL (from untrusted part to trusted part) and OCALL (from trusted part to untrusted part). The specific logic in each manipulation is as follow:

- Function Call: uses the *memcpy(dst_buf,src_buf,0)* function, basically means a void function;
- System Call: uses one of most simple system call function, which is *GetPriorityClass()*;
- ECALL: calls a void function from untrusted part to trusted part;
- OCALL: same as ECALL, call the void function from the trusted part to the untrusted part;

For each call, we choose the loops from 2000 times to 4000 times, measuring the specifically spent time in both hardware way and software way. After that, we calculate the results and get an average time of each call. The result (see Figure 2) shows that the function call has the lowest cycles per operation, which also mean the lowest overhead. The system call takes 200 cycles per operation. However, even though the ECALL and OCALL are just two void functions, the experiment

uncovers their cycles per operation are extremely higher than system call and function call, which is around 35 times of the normal system call. From this result, we can infer the cost of crossing the enclave is not as low as we can ignore.

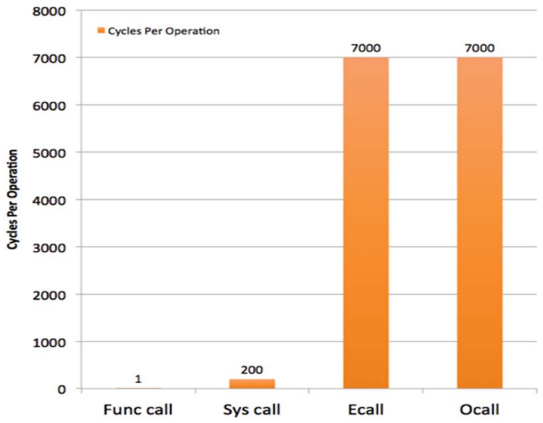

Fig. 2. Overhead comparison between different call

B. Performance Cost: enclave crossing in a real application

In order to make our benchmark more persuasive and forceful, we choose the Bzip2 library from Standard Performance Evaluation Corporation (SPEC) 2016 to measure the overhead caused by ECALL. SPEC, as for a standard benchmark suite for CPU and some other hardware [13], it is totally suitable to evaluate the performance of SGX. We conduct an experiment of inserting randomly ECALL instruction with different length of the coding gap. For instance, we first calculate the origin time of running the Bizp2 benchmark without the ECALL instruction, which represents the infinity of our ECALL interval. Then we insert one ECALL manipulation with every 16000 lines code and calculate the whole spending time of running Bzip2. With the number of the coding gap gradually decreasing, we obtain more result to help us analyze the connection between the time overhead and the frequency to call ECALL in practical code. The experiment result (see Figure 3) highlights the overhead time get increasing sharply in the real application with the growth frequency of the ECALL instruction.

With the question of whether the interval of ECALL instructions is over-estimated, we check the practical frequency of the system call instructions settled in the typical workload. From Livio Soares's works [14] (see Table 1), we know that the interval of system call instructions in different server software fluctuates from 2445 to 12534. With the current SGX pattern, each system call represents one ECALL manipulation, therefore we can assure the interval setting of ECALL instructions in our experiment totally matches the authentic situation and the interval is not over-estimated.

C. Cost of Memory Access

As we inferred before, the enclave memory keeps encrypted as it running, therefore the ability of SGX to get memory access and to dynamically allocate memory allocation is highly questionable. To figure out how specific SGX influences the memory access, we make another benchmark measure the memory copy speed in the different region. The result shows (see Table 2) the fastest speed of copy memory region is outside enclave, and the lowest speed is inside the enclave.

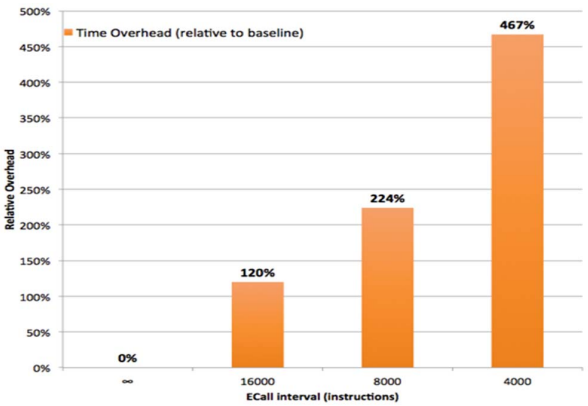

Fig. 3. Bizp2 benchmark from SPEC CPU 2016 with randomly insert ECall

TABLE I. FLEXIBLE SYSTEM CALL SCHEDULE WITH DIFFERENT WORKLOAD

Workload(server)	Instructions per Syscall
DNSbench(BIND)	2445
ApacheBench(Apache)	3368
Sysbench(MySQL)	12435

TABLE II. COST OF MEMORY ACCESS

Memory Copy Mode	Different region		
	Outside Enclave	*Cross Enclave*	*Inside Enclave*
Band Width (GB/s)	4.0	2.6	1.2

To figure out how much overhead caused by SGX when we allocate memory inside the enclave, we make the experiment of comparing the speed of *malloc()* and *free()* function which inside and outside the enclave. By setting different sizes of the allocated memories with the *malloc()* function, we get the different cycles per operation which can represent the actual overhead happened inside the enclave(see Figure 4). From the experiment result, it is intuitive to show the cycle per operation inside the enclave gets extremely more increased through the increasing size of allocated memory compared with that outside the enclave. After the malloc size gets 64 bytes, the cycles per operation inside the enclave turn to almost three times larger than which outside the enclave. Through the experiment result, we can conclude that the overhead indeed exists when we allocate memory inside the enclave. As we analyzed before, the reason of the overhead is probably because the enclave memory keeps encrypted as it running. The encrypting may highly impact the ability of SGX to have memory access and dynamic memory allocation.

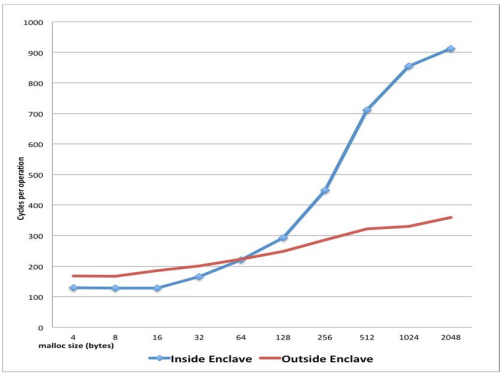

Fig. 4. The overhead time of *malloc()* inside and outside enclave

IV. CONCLUSION

SGX is indeed a new break-through to thoroughly solve today's cloud security problem in a hardware-assisted approach. Compared with other current solution, SGX has some great temptations like the guaranteed high security, low price and also efficient deployment. However, the result of our study unveils a number of critical issues with SGX programming mode and performance. The result also allows for identifying several research problems, whose timely solution could significantly enhance the chances for SGX to turn into a mainstream strategy for cloud security. With more understanding of current SGX mode and the result of our benchmark experiment, we can conclude that there are still the following challenges SGX need to overcome:

- In terms of performance, SGX needs minimal the runtime overhead. There is huge performance overhead when we use the SGX in our system. As a reason of lacking the ability to directly manipulate system call inside the enclave, we can only cross outside of the enclave to made the system call, which will arise more unexpected overhead. From our result, those overheads are huge enough to affect the performance of cloud applications. Besides, the memory copy and malloc speed inside the enclave is much slower than outside the enclave. The result shows the cost of memory accessing by the enclave is also one crucial factor of the largest performance costs.
- In terms of development, SGX should minimal the engineering effort. With the current SGX mode, any application's code ought to be divided into two parts, which are trusted code component and untrusted code component. This requires the application designers have to redesign or refactor the application to fit the guidelines. However, the requirement is totally not practical, especially for some big server applications like Apache which has a huge amount of codes. Rebuilding the application structure to obtain the guarantee of security is clearly not an efficient way.

V. FUTURE WORK

As for one of the new solutions to resolve the cloud security problem, SGX brings us a new perspective and strategy to reconsider the way to protect cloud security. Considering the huge performance cost and the complexity of dividing the structure of application into several parts, SGX seems not applicative for today's cloud environment. However, some irreplaceable properties that SGX owns make us still hold the faith that it would turn into a mainstream strategy for cloud security in the near future. Our future work will focus on finding an effective solution to thoroughly resolve the previous challenges we proposed. We hope the experiments in this work would also motivate more research on SGX performance for the cloud environment.

ACKNOWLEDGMENT

This work was supported by the National High-tech R&D Program of China (Grant No. SS2015AA020102), the 1000-Talent program, Tsinghua University Initiative Scientific Research Program.

References

[1] Baumann, Andrew, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven." *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015): 8.

[2] Adrian Chen, "Google Engineer Stalked Teens, Spied on Chats". http://gawker.com/5637234/gcreep-google-engineer-stalked-teens-spied-on-chats.

[3] Germany Tackles Tax Evasion. http://online.wsj.com/news/articles/SB10001424052748704197104575051480386248538.

[4] Subashini, Subashini, and Veeraruna Kavitha. "A survey on security issues in service delivery models of cloud computing." *Journal of network and computer applications* 34.1 (2011): 1-11.

[5] Mathew, Sajee. "Overview of amazon web services." *Amazon Whitepapers*(2014).

[6] Hofmann, Owen S., et al. "Inktag: Secure applications on an untrusted operating system." *ACM SIGARCH Computer Architecture News*. Vol. 41. No. 1. ACM, 2013.

[7] Zhang, Fengzhe, et al. "CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization." *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011.

[8] Bajaj, Sumeet, and Radu Sion. "TrustedDB: A trusted hardware-based database with privacy and data confidentiality." *IEEE Transactions on Knowledge and Data Engineering* 26.3 (2014): 752-765.

[9] Software Guard Extensions Programming Reference, Rev. 2. Intel Corp., Oct. 2014. Ref. #329298-002.

[10] Costan, Victor, and Srinivas Devadas. *Intel sgx explained*. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint. iacr. org/2016/086.

[11] Intel Corporation，Intel® Software Guard Extensions SDK User's Guide. 2015 Intel Corporation，https://software.intel.com/en-us/sgx

[12] Surenthat Selvaraj.(Intel) "Overview of Intel® Software Guard Extensions Instructions and Data Structure", June 10, 2016 https://software.intel.com/en-us/blogs/2016/06/10/overview-of-intel-software-guard-extensions-instructions-and-data-structures

[13] Henning, John L. "SPEC CPU suite growth: an historical perspective." *ACM SIGARCH Computer Architecture News* 35.1 (2007): 65-68.

[14] Soares, Livio, and Michael Stumm. "FlexSC: flexible system call scheduling with exception-less system calls." *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010.

[15] Hoekstra, Matthew, et al. "Using innovative instructions to create trustworthy software solutions." *HASP@ ISCA*. 2013.