VoltJockey: Breaking SGX by Software-Controlled Voltage-Induced Hardware Faults

Pengfei Qiu[1,2,3], Dongsheng Wang[1,2], Yongqiang Lyu[2*], Gang Qu[3]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China.
[2]Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, China.
[3]Dept. of Electrical and Computer Engineering & Institute for Systems Research, Univ. of Maryland, College Park, USA.
qpf15@mails.tsinghua.edu.cn, {wds, luyq}@mail.tsinghua.edu.cn, gangqu@umd.edu

***Abstract*—Intel software-guard extensions (SGX) allows applications to run in a trusted space (enclave), which provides a highly secure primitive for the running codes and data. Most state-of-the-art attacks on SGX either exploit software vulnerabilities in the enclave or utilize side channels. In this study, we propose the first fault injection attack to break SGX by using voltage-induced hardware faults. This attack is completely controlled by software. It does not require the availability of any software vulnerability.**

Our proposed attack targets multi-core processors where dynamic voltage and frequency scaling (DVFS) is equipped, which covers most of the commercial processors including those by Intel and ARM. We follow the basic principle for frequency and voltage based fault injection attacks which adjust the frequency or voltage levels deliberately to create hardware faults. We have demonstrated that the software-controlled voltage glitches are effective to fault ARM processors in the previous work. However, to the best of our knowledge, this is the first realization of such attacks on SGX. More specifically, we develop a kernel module to schedule frequency and voltage for Intel processors through their model-specific registers (MSR). We firstly utilize the module to furnish the processor a transient low voltage with controlled timing to inject a temporal fault into the target location of the program running on SGX. Then we perform differential fault analysis on the outputs before and after the injection of faults. This allows us to obtain the SGX-protected confidential data such as encryption keys, with which we can access program and data in the enclave or run any application in the enclave. For demonstration, we successfully deploy the proposed attack to extract the key of an AES executed in the SGX enclave.

***Index Terms*—Intel SGX, voltage manipulation, hardware fault injection, DVFS, software-controlled hardware attack**

I. Introduction

Intel software-guard extensions (SGX) extends the hardware and software architectures to provide a secure execution environment (known as *enclave*) to separate the trusted and untrusted sections of applications. The trusted base of SGX is only bound to CPU, which means that the enclave-private data are explicitly out of reach from the other software running at any privilege level, including, operating system (OS), virtual machine monitor (VMM), basic input output system (BIOS). Industry and academia have quickly adopted SGX in a wide variety of security-related applications such as secure key generation, secure data processing, secure encryption and decryption, and secure authentications [1].

SGX utilizes the hardware-assured security methodologies to protect the confidentiality, integrity, and availability of codes and data in the enclave, which strongly prevents the trusted data from being stolen. Existing attacks on SGX rely on the application-specific information leakages from either software vulnerabilities or side channels [2]. Well-developed enclaves are secure against these attacks because the information leakage can be effectively addressed [2]. In this paper, we propose to break SGX by hardware fault injection via voltage manipulation (another powerful exploitation of *VoltJockey* [3]) on multi-core processors where dynamic voltage and frequency scaling (DVFS) [4] is enabled. The proposed attack relies on the processor hardware features and all the attack steps are implemented with software. It is a new type of attack on SGX because it does not exploit any software vulnerability.

DVFS achieves energy saving by dynamically adjusting the frequency and voltage of processor cores per the workloads and performance demands [5]. It has been adopted by almost all modern systems including Intel and ARM processors. Processor's speed or clock frequency is supported by the supply voltage. A stable clock frequency is vital for the system's execution and can be provided by a range of voltages. However, when the voltage goes out of this range, the desired clock frequency may not be maintained, thus hardware faults could occur. This makes it possible for attackers to collect the information of their deliberately injected errors and extract sensitive data by methods like differential fault analysis.

The software-controlled frequency-induced hardware faults attack has been applied in the CLKscrew attack [6] that breaks ARM TrustZone by overclocking. However, SGX is resilient to this attack because, unlike ARM processors, most of the commercial low power Intel CPUs do not support overclocking. The effectiveness of such voltage-induced hardware faults on ARM processors has also been verified on our previous work [3]. In this paper, we demonstrate that the voltage-induced hardware fault attack is feasible to break SGX. More specifically, we have made the following contributions:

1) We identify that the voltage and frequency of multi-core Intel processors can be changed by modifying the model specific registers (MSR) with kernel software from any

*Corresponding author

978-1-7281-3544-1/19/$31.00 ©2019 IEEE

core. We then develop a kernel module that can configure the MSR to precisely inject faults by lowing voltage.
2) We propose a hardware fault attack based on our developed kernel module. To the best of our knowledge, unlike the existing attacks on SGX, this is the first fault injection attack that does not rely on any software vulnerability.
3) We apply the proposed attack on a commercial Intel processor with AES running in the enclave and successfully obtain the encryption key.

II. Preliminaries and Related Work

In this section, we provide the background on Intel SGX and DVFS as well as frequency and voltage management of Intel processors. Besides, we present the relevant work on frequency and voltage-based fault injection attacks.

A. Intel SGX

Intel SGX enforces the confidentiality, integrity, and availability of enclave-private data with a minimal trusted base that includes only the processor. The memory pages of codes, data, and structures of an enclave are located in the enclave page cache (EPC). SGX provides a structure of enclave page cache metadata (EPCM) to manage every page of EPC, which ensures that the enclave-private memory can never be accessed from outside but the untrusted section remains in charge of the enclave programs. Moreover, SGX provides local and remote attestations for providing secure channels between local enclaves or remote enclaves.

SGX enhances the security of software to a remarkable extent and largely builds an obstacle for normal attacks. The current attacks on SGX mainly focus on the software vulnerabilities of trusted instructions or side channels, which can be addressed by carefully developing the trusted programs and deliberately organizing the secret data [2].

B. DVFS

The amount of energy consumed in the processor is the integral of instantaneous power over time, where the total power mainly consists of dynamic power and static power. During run time, the dynamic power is proportional to the clock frequency and operating voltage[1] [4]. Consequently, the voltage and frequency will help in reducing energy dissipation. DVFS dynamically adjusts the frequency and voltage of processor cores to maximize energy saving. Practically, commercial processors store the discrete operational frequencies in the so-called *frequency table*. The OS provides driver commands to configure the CPU frequency manually but no system command is available for regulating the voltage directly.

C. Frequency and voltage management of Intel processors

For frequency controlling, the outputs of the hardware frequency regulator are designed to be multiples of a base frequency and the privileged DVFS driver simultaneously changes the frequency in terms of processor states. For voltage management, the output of the hardware voltage regulator will also be changed when DVFS driver alters the frequency. However, not all of the Intel processors support overclocking[2]. Therefore, these processors will not be affected by the overclocking-based fault attacks such as CLKscrew [6].

Software cannot configure the two hardware regulators to alter their outputs directly. Instead, Intel provides MSR for privileged software to set them. More specifically, the register of `IA32_PERF_CTL` represents the turbo ratio of frequency. However, there is no public document about the register that defines the processor voltage. Although every core has its own MSR, most Intel processors are designed to distribute the same frequency and voltage to all the cores.

D. Frequency and Voltage-based Fault Injection Attacks

The main principle behind the frequency-based fault attacks [7-9] is that the expected output of a circuit may not be ready for the use in next unit if the frequency is too high. However, most of them change the frequency with auxiliary hardware. Tang et al. proposed the Clkscrew [6] that induces frequency-based faults with software, which has been conducted to break the ARM devices. However, the processor is still required to enable the overclocking technique. Unlike this, our proposed attack mechanism reduces the processor voltage with software instructions to induce faults. In the example of Intel processors, we exploit the MSR directly to make this happen. Since we do not overclock, frequency locking method will not be effective against our proposed attacks.

Like frequency, voltage is also one of the most significant elements to provide an appropriate working environment for processor; changing the voltage to create faults has attracted a lot of attentions [10-13]. Compared to the existing voltage-based fault injection approaches, our attack can be launched by software only without using any assistant hardware units. Besides, we experimentally verify the attack on a commercial Intel processor and demonstrate that the trusted execution environment may not be secure because we can successfully infer the encryption key of AES performed in the SGX.

III. Overview of the Software-controlled Voltage-induced Hardware Fault Attack

Fig. 1 illustrates the overview of our attack approach. We consider a multi-core system, where an attacker can launch attack on one core, which we will refer to as the attacker core, by creating a short-lived low voltage. This might trigger hardware faults on other cores, which we will call victim cores, such that incorrect outputs will be produced.

A. Assumption

We assume that (1) the target processor has multiple cores and is DVFS-enabled, and (2) the attacker has permission to set the processor's voltage and frequency. The goal of the attacker is to gain access to or steal data from a victim core.

[1]Formally, the dynamic power P_t at time t is: $P_t \propto V_t^2 F_t C$, where V_t is the voltage, F_t is the frequency, and C is the capacitative load.

[2]The Intel processors whose version number ended with *K* enable the overclocking technique. However, the most deployed ultra-low-voltage processors cannot be overclocked.

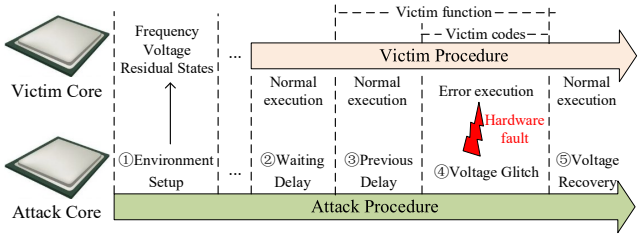

Fig. 1. Overview of our voltage-induced fault attack. Top: victim procedure on the victim core. Bottom: attacker procedure on the attacker core.

Assumption (1) is actually true for majority of the current processors as DVFS is one of the most effective methods for energy saving. Normally, the voltage and frequency on such processors can be adjusted by software. Assumption (2) may require the attacker to have some root privilege on the system. This does not violate the trusted base of SGX and there are many legal or illegal ways to gain such access [14].

B. Attacker Procedure

The attacker procedure is performed on the attacker core with the goal of introducing hardware faults into the victim cores. This can be achieved in the following five steps.

(a) **Environment setup**. An adequate voltage glitch environment should first be prepared before carrying out the attack. This includes configuring the processor with a fixed frequency, assigning a proper voltage to support the frequency, and clearing the residual states that may affect the attack (which will be elaborated in Section IV-A).

(b) **Waiting delay**. It is unknown when the victim function would be executed. Therefore, the attacker procedure needs to wait for the target function to be invoked before triggering desired hardware faults. We call this as the *waiting delay*.

(c) **Previous delay**. Normally, the function into which an attacker would inject errors is a small portion of the victim procedure. Therefore, the fault injection points should be carefully controlled to ensure that the faults will not have a large impact on other portion of the victim function. We refer to this as *previous delay* and will discuss it in Section IV-E.

(d) **Voltage glitch**. Hardware faults are a result of applying voltage lower than the proper value for a period of time. It is crucial to regulate the low voltage and its duration in order to create glitch and induce hardware faults.

(e) **Voltage recovery**. Once the attacker has incurred the desired hardware faults, he needs to restore the voltage to the original value so the victim procedure can resume its normal execution. This also helps the attacker from being discovered.

C. Attacker parameters

To perform the above attacker procedure, an attacker needs to determine the following six important parameters.

$$F_{fault} = \{F, V_l, V_b, T_w, T_p, T_d\}$$

where F represents the processor's frequency, V_l is the glitch voltage that induces hardware faults, V_b is the voltage assigned to maintain the frequency F, T_w is the waiting time before the victim function is executed (*waiting delay*), T_p is the waiting time before the victim codes are invoked (*previous delay*), and T_d represents the duration time that the glitch voltage should be kept.

D. Victim Procedure

The top half of Fig. 1 shows the execution on the victim core at each step of the attacker procedure. Once the victim function starts running, the period of "waiting delay" is over and the "previous delay" begins till the victim codes are executed. Then a short period of hardware fault should be incurred by the attacker before the core resumes its normal execution.

Any program running on the victim core could become the victim procedure. However, the attackers will be more interested in programs that can lead to malicious actions, such as stealing sensitive data or modifying the output. In SGX, encryption methods have been adopted to ensure data seal, page swapping, and attestations [1]. Therefore, the encryption function could be a good candidate for the victim procedure. Many encryption schemes, including RSA, DES, ECC, SMS4, and Grain family [15], have been broken by fault analysis. Our contribution is the discovery of a practical solution to create hardware faults during the encryption/decryption operation to facilitate breaking SGX by differential fault analysis.

IV. Implementation of the Attacker Procedure

In this section, we present solutions and experiment probings for the challenges in implementing the attacker procedure. Without loss of generality, we consider a victim AES of a 128-bit block cipher, which encompasses 10 rounds operations. The experiment device is a DELL XPS laptop that has an i7-8550U CPU, 8G memories, and 8 cores; the OS is Ubuntu 16.04.

A. Clearing Residual States

It is crucial to keep the execution time of the victim instructions stable for the accurate identification of the fault injection points The execution time of instructions may be affected by the system's residual states from three aspects, namely, cache layouts, branch prediction table, and system interrupts. We first flush the cache memory to clear the irrespective data and then run the victim procedure several times to fill the cache memory with the victim procedure-correspondent codes and data. The branch prediction table will also be highly related to the victim procedure. Moreover, we block the interrupt requests (IRQ) that target the victim core.

B. Voltage and frequency Manipulation

As presented in section II-C, Intel does not publish any documents about the MSR that controls the voltage. However, we find that fortunately we can change the processor voltage by setting the voltage offset with the MSR[3]. There are two available technologies to assign the processor frequency, utilizing the DVFS commands or configuring the MSR directly.

[3]This has been confirmed during our conversation with the Intel Product Security Incident Response Team.

The frequency is unchanged in the whole attack, we choose to stabilize the frequency by invoking the driver commands.

C. Attack Kernel

Although the values of MSR can be reset by driver commands provided by the *msr* module, it is very time consuming and may not generate transient glitch voltages. In order to accurately maintain the duration time of the glitch voltage for better fault injection precision, we implement a kernel module (the attack kernel) to achieve the above 5-step attack procedure. The module first eliminates the residual states and initiates the processor with a safe voltage. Next, it waits for the victim function and the appropriate injection points, and then conducts the low voltage glitches. Finally, the working environment is reset to the original one. The module does not need to be integrated into the kernel image, and it can be dynamically loaded while attacking.

D. Voltage Selection

In order to quickly locate the available attack voltages, we first measure the default voltages that are related to frequencies in the frequency table through MSR and illustrate them as the black curve of Fig. 2. Then, we obtain the boundary voltages that will ensure the system is stable when the processor is idle. Finally, we run a computational task that invokes several threads on different cores to ensure the processor is busy and observe the boundary voltages again. As displayed in Fig. 2, the boundary voltages when the processor is vacant are visibly higher than that when it is busy. For a certain frequency, voltages between the black line and blue line are safe, voltages between the blue line and red line are likely to induce hardware faults, and voltages below the red line can cause hardware faults with a high probability. Therefore, we can choose the voltages below the red line as the attack voltages.

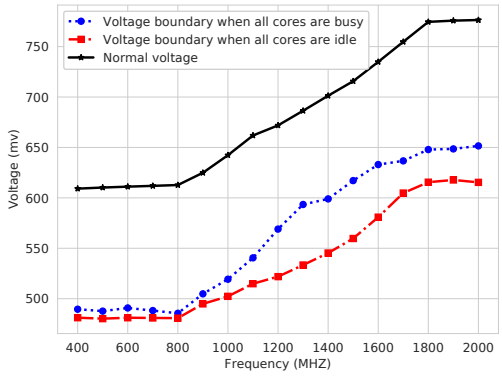

Fig. 2. The manipulable voltages for Intel i7-8550U processor.

E. Previous Delay

The previous delay measures when the hardware faults will be induced, which is mainly decided by two factors: 1) the program instructions of the victim function; 2) the processor

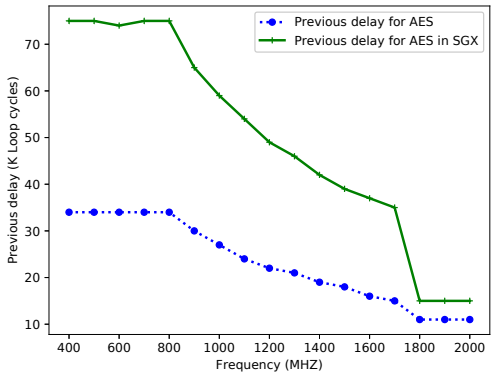

Fig. 3. The previous delay for AES with different frequencies, which is almost stable in low or high frequencies and reduces with the frequency augments from $900MHZ$ to $1700MHZ$.

frequency. The processor voltage has negligible effects on the previous delay as the processor speed is mainly determined by the frequency. For a given victim function, especially a public encryption function, its instructions are commonly known. Hence it is possible to extract the previous delay in advance. In this study, we implement a 128-bit AES encryption function based on S-box and measure the `NOP` loop cycles from the start of AES to the `MixColumn` operation of the seventh round with different frequencies, which are presented in Fig. 3. Generally speaking, a `NOP` operation does nothing except occupying a cycle, it is an excellent operation for timing in the clock accuracy. The previous delay is stable when the frequency is no more than $800MHZ$ or no less than $1800MHZ$. Besides, the previous delay is negatively correlated to the frequency among $900MHZ$ and $1700MHZ$. In general, the previous delay should always decrease with respect to increasing frequency. We suppose some mechanisms may be enforced to balance the performance when the frequency is low or high.

F. Improving the attack performance

Although the attackers can straightway analyze the faulty output and error-free output to infer the sensitive data, they would need more efforts as the hardware faults might not exactly result in the expected data modifications. The attacker procedure cannot directly read the data of the victim procedure as they are different processes. In this study, we speculate the changes of the state matrix driven by the hardware faults with cache side-channels. There are several side-channel technologies available, for instance, *prime+probe*, *flush+reload*, and *evict+reload* [16]. Brasser et al. verified that the cache side channel attacks are practical for SGX [16].

V. Breaking AES in SGX

In this section, we first present the differential fault analysis on AES with a single fault and then speculate the encryption key of the AES performed in the normal execution environment and SGX with our proposed attack method, respectively.

A. Differential Fault Analysis on AES with a Single Fault

As a widely employed symmetric encryption method, it is impractically time-expensive to brute force search for the encryption key of AES, particularly when the key is sufficiently long. However, a lot of researches have shown that the candidate key space can be significantly reduced when certain errors occur during the execution [17-19]. In this study, we adopt the method proposed by Tunstall et al. [18], which extract the AES key when a single random byte fault is introduced into the input of the $(R-2)$th (R is the magnitude of the total rounds, it is 10 in this work) round.

A single-byte error in the input of the eighth round will generate four byte-errors to the input of the ninth round. Similarly, the four byte-errors will be exactly propagated to the complete state matrix of the tenth round. With the equations of the encrypted data and the input, the attacker can create 2^{32} key assumptions for the 128-bit key. This, combined with the fact that the generation process for round keys is reversible, can reduce the number of key conjectures to 2^8 [18]. We combine the key hypotheses for different plaintext-ciphertext pairs to further reduce the search space.

B. Normal Execution Environment Attack

We select different attack parameters to analyze the key of AES that is performed in the normal execution environment. Because the operations on plaintext in every round is fixed, we do not consider the T_w in this attack. When inducing hardware faults into the victim core, the other cores are also suffering from hardware faults as the voltage manipulation is conducted on all cores. When the induced hardware faults influence the kernel software, a kernel panic may occur if the OS cannot recover quickly. The typical results of a kernel panic are system freeze or reboot. Therefore, some attack attempts may fail. According to the experiment tests, when the attack parameter is $\{F = 1.4GHZ, V_l = 0.55V, V_b = 0.754V, T_w = 0, T_p = 19585, T_d = 5200\}$, we successfully inject 10 errors into the input of the eighth round among 500 attempts and 3 of them are one-byte errors.

C. Secure World Attack

SGX shares physical cores with the normal execution environment. We verify that the processor voltage can be altered with instructions executed in the normal execution environment on any core. Therefore, SGX is also vulnerable to the proposed attack although it is isolated.

1) Inserting AES into SGX: Intel provides the SGX driver, software development kit (SDK), and platform software for programmers to develop SGX-based software. We implement a SGX-based program in which the AES encryption function is embedded and bind the program to the victim core in the untrusted instructions,

2) Exploring the T_w: The AES is performed after the untrusted instructions invoke the trusted portion, therefore, we can find the precise value of T_w. Similarly, because T_p is determined by the processor frequency for a fixed program, we can monitor the execution time from the beginning of the victim procedure to the start of AES with different processor frequencies. The experimental results are listed in Table I, where we see that T_w has similar characteristics as T_p.

3) Attack parameters: Although some security mechanisms are enforced in SGX to check every operation, most of them are hardware-based and have negligible influences on the execution time of the trusted instructions once they have been loaded into the EPC. Therefore, the attack parameters in the normal world will still work for SGX.

4) Performing attack: We utilize the attack parameter exploited in the normal execution environment with the T_w is $1.607 * 10^6$ to steal the encryption key. With 500 attempts, we get four different faulty outputs that are influenced by a single error in the input of the eighth round. When the error location is 8, the differential fault analysis method takes about 926 seconds to generate 3623 key hypotheses. Besides, it spends around 1121 seconds to generate 4248 key candidates when the fault location is 13. We compare the two key hypotheses and they have only one value in common, which turns out to be the encryption key.

VI. Countermeasures and Future Work

In this section, we present the possible countermeasures for mitigating our attack from hardware and software views. Besides, we also discuss some extensions of this work.

A. Hardware-based Countermeasures

1) Limiting the processor voltage: In the proposed attack, we induce controllable hardware faults by adjusting the processor voltage with software, therefore, abolishing the interfaces for software instructions to alter the voltage is a useful countermeasure. The simplest method is to power the processor with a fixed voltage. In this defense, the voltage offset MSR should also be prohibited. Automatically adjusting the voltage with supererogatory hardware units to support the current frequency is also an available method. Those approaches are effective but will have drawbacks in resource and performance as well as limit the effectiveness of DVFS.

2) Designing secure cores for SGX: A precondition for the attack is that the adversaries should be equipped with a high privilege. Therefore, attacking SGX will be the most common application. We can design isolated secure cores for SGX to prevent the attack. However, this will be resource-expensive and increase the hardware cost. Besides, this requires that the hardware regulator and corresponding drivers for secure cores be designed independently.

B. Software-based Countermeasures

The processor voltage and frequency can be read in software. Therefore, we can develop a SGX-based trusted application to continually monitor the processor and trigger protection mechanism if it finds the voltage is out of the legal ranges for the current frequency. However, if the monitor frequency is high, the processor usage will be high and more energy will be dissipated. Otherwise, attackers still have the possibility to find feasible injection sites as the glitch duration is very short in the attacks.

TABLE I
THE WAITING TIME FOR AES WITH DIFFERENT FREQUENCIES

Frequency (MHZ)	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Waiting time(10^6 loop cycles)	2.783	2.781	2.809	2.809	2.804	2.474	2.236	2.040	1.855	1.732	1.607	1.488	1.399	1.316	0.696	0.695	0.696

C. Future Extensions

The attack might also be extended to the following scenarios.

1) Attacking other cryptosystems: Besides AES, our approach can be applied to attack other well-known cryptosystems that have been mathematically proved to be vulnerable to differential fault analysis including RSA, DES, 3G-SNOW, ECC, SMS4, and Grain family [15].

2) Creating vulnerabilities for software attacks: The voltage-induced hardware faults can potentially help the attackers to modify the execution data or break the control flow of a program. For example, the router would send the Internet package to a malicious destination if the package was modified at packaging; the control flow might be changed if the targets of direct or indirect jumps were altered.

VII. CONCLUSIONS

SGX extends the hardware and software architectures of Intel processors to supply an isolated secure execution environment to protect trusted codes and data. In this paper, we propose a software-controlled voltage-based hardware fault attack for SGX, which exploits the voltage management vulnerability of DVFS technique. We verify that the voltage and frequency of the Intel processors can be governed by configuring the MSR registers with software and develop a kernel module to alter the core frequency and voltage. The processor voltage is reduced dramatically at a specific moment and will be kept for a certain amount of time. Consequently, controllable hardware faults will be introduced into the victim program and result in faulty outputs. Experiments show that the attack method can successfully acquire the encryption key of the AES protected by SGX, which shows a comparable efficiency to other attacks in obtaining SGX-guarded credentials. At the end of the paper, we also analyze and discuss the possible countermeasures and potential extensions of this work.

VIII. ACKNOWLEDGMENTS

We thank the anonymous reviewers and the Intel Product Security Incident Response Team for their valuable comments and discussion. This work was supported in part by the National Key Research and Development Plan of China under Grant No. 2016YFB1000303 and the Guangdong Province Key Project of Science and Technology under Grant No. 2018B010115002.

REFERENCES

[1] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[2] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution," in *27th USENIX Security Symposium*, 2018, pp. 991–1008.

[3] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies," in *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS'19)*. London, United Kingdom: ACM, 2019.

[4] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1702–1714, Dec 1999.

[5] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ser. ISLPED '04. New York, NY, USA: ACM, 2004, pp. 174–179.

[6] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1057–1074.

[7] J. Delvaux and I. Verbauwhede, "Fault injection modeling attacks on 65 nm arbiter and ro sum pufs via environmental changes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 6, pp. 1701–1713, June 2014.

[8] Q. Wang, A. Wang, L. Wu, G. Qu, and G. Zhang, "Template attack on masking aes based on fault sensitivity analysis," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, vol. 00, May 2015, pp. 96–99.

[9] D. Saha, D. Mukhopadhyay, and D. Roy Chowdhury, "A diagonal fault attack on the advanced encryption standard." *IACR Cryptology ePrint Archive*, vol. 2009, p. 581, 01 2009.

[10] N. Selmane, S. Guilley, and J. Danger, "Practical setup time violation attacks on aes," in *2008 Seventh European Dependable Computing Conference*, May 2008, pp. 91–96.

[11] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Low voltage fault attacks to aes," in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2010, pp. 7–12.

[12] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the rsa cryptosystem," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sept 2009, pp. 23–31.

[13] A. Barenghi, G. M. Bertoni, L. Breveglieri, and G. Pelosi, "A fault induction technique based on voltage underfeeding with application to attacks against aes and rsa," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1864–1878, 2013.

[14] S. Niu, J. Mo, Z. Zhang, and Z. Lv, "Overview of linux vulnerabilities," in *2nd International Conference on Soft Computing in Information Communication Technology*. Atlantis Press, 2014/05. [Online]. Available: https://doi.org/10.2991/scict-14.2014.55

[15] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 513–525.

[16] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:sgx cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[17] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Fault attack on aes with single-bit induced faults," in *2010 Sixth International Conference on Information Assurance and Security*, Aug 2010, pp. 167–172.

[18] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, C. A. Ardagna and J. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 224–233.

[19] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on a.e.s," in *Applied Cryptography and Network Security*, J. Zhou, M. Yung, and Y. Han, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 293–306.