# DRAMD: Detect Advanced DRAM-based Stealthy Communication Channels with Neural Networks

Zhiyuan Lv *‡, Youjian Zhao *‡, Chao Zhang †‡§, Haibin Li *‡
*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
†Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China
‡Beijing National Research Center for Information Science and Technology(BNRist), Beijing 100084, China

***Abstract*—Shared resources facilitate stealthy communication channels, including side channels and covert channels, which greatly endanger the information security, even in cloud environments. As a commonly shared resource, DRAM memory also serves as a source of stealthy channels. Existing solutions rely on two common features of DRAM-based channels, i.e., high *cache miss* and high *bank locality*, to detect the existence of such channels. However, such solutions could be defeated.**

**In this paper, we point out the weakness of existing detection solutions by demonstrating a new advanced DRAM-based channel, which utilizes the hardware Intel SGX to conceal *cache miss* and *bank locality*. Further, we propose a novel neural network based solution *DRAMD* to detect such advanced stealthy channels. DRAMD uses hardware performance counters to track not only *cache miss* events that are used by existing solutions, but also counts of *branches and instructions executed, as well as branch misses*. Then DRAMD utilizes neural networks to model the access patterns of different applications and therefore detects potential stealthy communication channels. Our evaluation shows that DRAMD achieves up to 99% precision with 100% recall. Furthermore, DRAMD introduces less than 5% performance overheads and negligible impacts on legacy applications.**

***Index Terms*—side channel, covert channel, SGX, neural network, countermeasures, DRAM**

## I. Introduction

DRAM-based side channel and covert channel attacks recently have attracted increasing attentions [1]. Since DRAM is shared by multiple cores and processors, DRAM-based side and covert channels can work across processors. Therefore, comparing with recent popular cache-based side channel [2] and covert channel [3] attacks, DRAM-based attacks pose more severe threats [4]. Although many countermeasures have been proposed to counter cache-based attacks [5]–[10], there are only a few straightforward methods to detect DRAM attacks [1], [4].

*DRAM attack and detection.* DRAM side and covert channel attacks [1] can break the security guarantee posed on information flow. Specifically, DRAM side channels can infer secret information processed by a victim application by measuring its DRAM usage patterns. DRAM covert channels enable covert communications between two cooperating entities. To measure the sender's/victim's DRAM usage, the attacker has to flush the cache to evict the memory being measured. Therefore, existing DRAM attack techniques in general have high *cache misses*. Moreover, DRAM attacks also have a high *bank locality*. Attackers need to perform repeated access of a few target DRAM banks, while regular programs will access many banks due to Intel's well-designed DRAM addressing algorithm [11]. Therefore, some studies have proposed using those two features to detect DRAM attacks [1], [4], [12].

*Advanced DRAM attacks.* However, existing detection solutions relying on the *cache miss* and *bank locality* features could be defeated, by concealing such features with certain mechanisms. In this paper, we demonstrate an advanced DRAM attack to defeat existing detection solutions by utilizing Intel Software Guard eXtensions (SGX) to conceal these two features. SGX is an x86 instruction set extension used to securely and confidentially run programs in isolated environments (i.e., enclaves) on systems potentially controlled by adversaries. Hence, cache misses and memory access in enclaves cannot be measured even by operating system (OS) [13], rendering existing detection solutions ineffective. Further, we also present another variant of attack, i.e., *one-row DRAM attack* which accesses only one DRAM row, able to further promote the success rate of the proposed SGX-based channel.

*Challenges.* To detect DRAM-based channels, especially the advanced attacks presented in this paper, there are several challenges that need to be addressed: (1) Detecting activities inside SGX. The SGX enclave is an isolated environment that cannot be accessed even by the OS. Therefore, providing an approach that can detect malicious activities inside SGX is challenging. (2) Detecting covert channels. Unlike side channels in which senders are victims, senders in covert channels are cooperative attackers, which do not cooperate with the defenders. Therefore, developing an approach that can detect both side and covert channel attacks without cooperation from the sender is challenging as well. Few work have focused on this issue. (3) Compatibility and universality. Since the target hardware, OS and software vary widely, providing a solution that can simultaneously protect both vulnerable legacy systems and future systems is a formidable challenge.

*Our solution.* In this paper, we present a lightweight and general software solution DRAMD to detect both traditional DRAM-based channels and the advanced SGX-based DRAM channels, including both side and covert channels. First, to detect activities in SGX, we propose a novel neural network based solution, which utilizes several hardware performance

§Chao Zhang is the corresponding author.

counters to classify different applications behaviors including stealthy channels. Although SGX shields its internal applications from these performance counters, it has to inter-operate with the OS to accomplish tasks. For example, the SGX application will be scheduled by the OS's process scheduler. Moreover, *different applications (no matter inside or outside SGX enclaves) present different performance counter patterns*. As a result, DRAMD is able to recognize SGX activities by monitoring the OS's performance counters. Second, DRAMD uses the host machine's hardware features for detection, without requiring the cooperation of the senders/applications. So it is able to detect both side and covert channels. Third, we design DRAMD as a lightweight software-only extension to the Linux kernel, which does not require new hardware support or OS modifications and can be deployed immediately, providing a good compatibility and universality.

In summary, this paper makes the following contributions.

- We point out the limitations of existing detection schemes and demonstrate an advanced attack that can bypass them. Specifically, our SGX-based attack does not have obvious abnormal features and cannot be monitored, even by OS kernel. The one-row DRAM attack variant further improves the success rate of the attack.
- We propose the first neural network based solution that can detect the internal behavior of SGX applications, and analyze the root causes.
- We present the first software-only system DRAMD, able to detect not only DRAM-based side channel attacks but also DRAM-based covert channel attacks. It introduces no changes to the hardware, OS or applications, and can be implemented in modern systems with few efforts.
- We present the full prototype implementation and extensive evaluation of the proposed approach. The evaluation result shows our solution is very effective.

## II. Background

### A. DRAM Organization

Modern DRAM is organized in a hierarchy of channels, DIMMs, ranks, and banks. Banks contain the actual memory arrays, which are organized into rows and columns.

**The row buffer.** Apart from the memory array, each bank also features a row buffer between the DRAM cells and the memory bus. From a high-level perspective, the row buffer behaves similarly to a directly mapped cache and stores an entire DRAM row. Requests to addresses in the currently active row are served directly from this buffer. If a different row needs to be accessed, then the currently active row is first closed (with a pre-charge command), then the new row is fetched (with a row-activate command) to the row buffer and served, and then this row becomes active. We call such an event a row conflict. Obviously, such a conflict leads to significantly higher access time compared to requests to the active row.

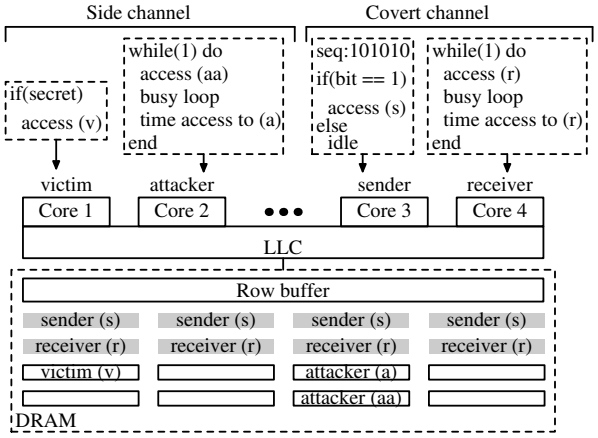


Fig. 1. DRAM-based side and covert channel attacks.

### B. DRAM-based Attacks

DRAM attacks exploit the physical architecture of modern DRAMs, in which each bank has a shared row buffer, even in multi-processor systems. While accesses to this buffer are fast, accesses to other memory rows in DRAM are much slower. This timing difference can be exploited to launch side and covert channels between two host applications. Fig. 1 shows an example of DRAM-based side and covert channel attacks.

**DRAM side channel:** As shown on the left side of Fig. 1, the victim process behaves differently based on the secret value. In order to leak the secret value (i.e., side channel), attackers will try to guess whether the victim memory has been accessed. To learn whether the victim process has accessed a virtual address (v), the adversary allocates two memory blocks (a) and (aa) that reside the same DRAM bank as (v). Further, the adversary makes (a) maps to the same row as (v), while (aa) does not. The adversary will perform a probing loop as follows: it first accesses the memory (aa), then waits for the victim action, and then measures the time of accessing memory (a).

When the secret value is 1, the memory (v) will be accessed, and the access time of (a) will be low since they share a same row which has been fetched to the row buffer. When the secret value is 0, the memory (v) will not be accessed during the probing period, and the access time of (a) will be high. As a result, the adversary could infer the secret of the victim.

**DRAM covert channel:** As shown on the right side of Fig. 1, the sender and the receiver occupy different rows in the same bank with the physical memory of (s) and (r). The sender will cooperatively send the secret to the receiver. When the secret to transmit is 1, it will access the memory (s), causing the receiver's time measurement high. Otherwise, it will suspend accessing the memory, causing the receiver's time measurement low. In this way, the sender could reliably send the secret to the receiver in a covert way.

### C. Intel SGX

SGX is a new set of x86 instructions introduced in the Skylake micro-architecture. SGX protects the execution of user programs by putting them in so-called enclaves. Only the application inside the enclave can access its own memory

region, and any other accesses to the enclave internal is blocked by the CPU or by the encryption imposed on the enclave data. Furthermore, not only the OS but also the hardware (including performance counters) cannot access or leak any information from the SGX enclaves[1].

Since SGX enforces this policy in hardware, enclaves do not need to rely on the security of the OS, the hardware, and even the cloud administrator. By performing sensitive computations (e.g., encryption) inside an enclave, one can effectively protect the application from traditional threats (e.g., malware), even if such malware has obtained kernel privileges. As a result, SGX provides a trusted execution environment and becomes widely adopted by cloud platforms, to provide stronger trust guarantee for users.

However, SGX could also be abused by attackers, e.g., to facilitate DRAM side and covert channel attacks, as shown in this paper.

### D. Modeling DRAM Stealthy Channels

In this subsection, we analyze the capacity of DRAM-based channels via information theory, proving that DRAM attacks must cause many cache misses.

*1) Transmission Error Rate:* Given a communication channel, the receiver may get a wrong value, i.e., different from the value sent by the cooperative sender or unwilling victim, due to communication noises.

In DRAM-based channels, there are also non-negligible noises causing transmission error. Hence, we assume the error rates in DRAM-base channels are $\xi$ and $\mu$ when transmitting 0 and 1 respectively, as shown in Fig. 2.

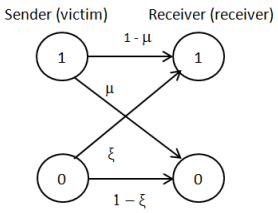



Fig. 2. DRAM side and covert channel attacks.

*2) Channel Capacity:* We denote the value sent by the sender as X and the value received by the receiver as Y.

Assume the probability of X as follows:

$$p(x_0) = p(X = 0) = p.$$
$$p(x_1) = p(X = 1) = 1 - p.$$

We can calculate the probability of Y as follows:

$$\begin{aligned} p(y_0) = p(Y = 0) &= p(x_0) \cdot (1 - \xi) + p(x_1) \cdot \mu \\ &= p \cdot (1 - \xi) + (1 - p) \cdot \mu \\ p(y_1) = p(Y = 1) &= p(x_0) \cdot \xi + p(x_1) \cdot (1 - \mu) \\ &= p \cdot \xi + (1 - p) \cdot (1 - \mu) \end{aligned}$$

Following Shannon's theorem [14], the entropy of Y is:

$$H(Y) = -\sum_j p(y_j) \cdot \log_2^{p(y_j)}$$

[1]Except for SGX side channels, e.g., the one discussed in our paper.

The conditional entropy of $(Y|X)$ is as follows:

$$\begin{aligned} H(Y|X) &= -\sum_{i,j} p(x_i) \cdot p(y_j|x_i) \cdot \log_2^{p(y_j|x_i)} \\ &= -\{p \cdot (1 - \xi) \cdot \log_2^{1-\xi} + p \cdot \xi \cdot \log_2^{\xi} \\ &\quad + (1 - p) \cdot \mu \cdot \log_2^{\mu} + (1 - p) \cdot (1 - \mu) \cdot \log_2^{1-\mu}\} \end{aligned}$$

Therefore, the mutual information of $(X, Y)$ is as follows:

$$I(Y|X) = H(Y) - H(Y|X)$$

Further, the channel capacity is as follows:

$$C = \max_p I(Y|X) \tag{1}$$

We could infer that, when the error rates $\xi$ and $\mu$ reach 0.5, the channel capacity becomes 0, i.e., no information can be reliably transmitted. When the error rates reach 0, the channel capacity could reach its maximum value. In practice, the communication parties will try to reduce the error rates.

*3) Error Rate Calculation:* In previous studies, covert channels have been commonly modeled as binary symmetric channels (BSC) [15]. In other words, the error rates of wrongly transmitting 0 and 1 in the channel are equal. However, these two error rates in DRAM-based channels are different.

**Error rate $\xi$ in covert channels (sending 0 but reading 1).** As shown in Fig. 1, in a covert channel, the receiver will repeatedly probe a row being monitored and measure the access time. To transmit a bit 0, the sender will suspend affecting the shared row buffer in the DRAM bank. The receiver will read a 0 since the estimated access time is low.

However, if *someone else accesses a row (different from the row being monitored) in this DRAM bank* during the probing period, the shared row buffer will become dirty. As a result, the receiver will read a 1 since the estimated access time is high, i.e., a transmission error.

Assuming the receiver's probing time interval is $\Delta t$, and *the probability of the DRAM bank being accessed* in each unit of time is $q_{t_1}, q_{t_2}, ..., q_{t_{\Delta t}}$, and the number of rows in the bank is $N$ ($N \geq 2^{14}$ for almost all DRAMs). Then, the error rate $\xi$, i.e., the probability that someone accesses the shared row buffer during the time interval $\Delta t$, is as follows:

$$\begin{aligned} \xi(\Delta t) &= \frac{N-1}{N}(1 - \prod_{i=1}^{\Delta t}(1 - q_{t_i})) \\ &\approx 1 - (1 - q_t)^{\Delta t}. \end{aligned}$$

where $q_t$ represents the distribution of $q_{t_1}, ..., q_{t_{\Delta t}}$. So, the error rate $\xi$ rapidly increases with the time interval $\Delta t$.

**Error rate $\mu$ in covert channels (sending 1 but reading 0).** Similarly, we could infer that, if someone else access the same row as the one being monitored during the probing period, the receiver will wrongly get a bit 0 when a bit 1 is sent.

Then, the error rate $\mu$, i.e., the probability that someone accesses the same row during the probing period, is as follows:

$$\begin{aligned} \mu(\Delta t) &= \frac{1}{N}(1 - \prod_{i=1}^{\Delta t}(1 - q_{t_i})) \\ &\approx \frac{1}{N}(1 - (1 - q_t)^{\Delta t}) \approx 0. \end{aligned}$$

So, $\mu$ is always maintained at approximately 0. Therefore, the channel capacity $C$ approaches to 0 along with the increase of the time interval $\Delta t$.

**Error rates in side channels.** Similarly, we could infer that, (1) the error rate $\xi$ in side channels equals to the error rate $\mu$ in covert channel, and (2) the error rate $\mu$ in side channels equals to the error rate $\xi$ in covert channel. The error rates also rapidly increase with the probing time interval.

*4) High Cache Misses of DRAM-based Channels:* Since the DRAM-based channels' error rates increase rapidly with the probing time interval, so in practice the attackers will reduce the probing time interval $\Delta t$ to reduce transmission errors. However, the DRAM probing process has to evict the cache and causes cache miss, in order to measure the DRAM access time in each probing period. So, a shorter probing period will yield higher cache misses.

In our experiments, we notice that attackers cannot reliably obtain any useful information if the probing time interval $\Delta t$ is greater than 1k CPU cycles. So, in practice, the probing time interval will be lower than 1K CPU cycles. Therefore, assuming the CPU frequency is at least 800 MHz, this attack will cause at least 800k cache misses in a second. Existing solutions could utilize this high cache miss feature to detect such channels.

## III. Existing Solution and Drawbacks

### A. Existing Detection Techniques

Existing detection techniques [1], [4], [12] rely on two common features of DRAM-based channels, i.e., high *cache miss* and high *bank locality*, to detect the existence of such channels.

*1) High cache miss feature:* As show in Sec. II-D, successful DRAM attack relies on repetitively accessing a few aggressor DRAM rows within a short time. Existing detection techniques make the observation that this fundamentally requires accesses to the aggressor rows to miss on all cache levels. This reveals two identifying characteristics of DRAM attack: high cache miss rate and high spatial locality of DRAM row accesses. This is in contrast to general memory access patterns where high locality results in high cache hit rates. As such it is straightforward to discriminate between DRAM attacks and non-malicious programs by looking at DRAM access patterns and rate.

*2) High bank locality feature:* Another property of DRAM attacks is high bank locality. A DRAM usually contains many banks, e.g., a DDR4 DRAM is composed of 32 banks [1]. In DRAM attacks, attackers need to repeatedly access only a few target DRAM banks, while non-malicious programs will access many banks due to Intel's well-designed DRAM addressing algorithm [11]. Therefore, this bank locality property can be used to discriminate between DRAM attacks and non-malicious programs.

### B. Breaking Current Detection Techniques

In this subsection, we show that these techniques are insufficient to guarantee protection from DRAM attacks. First, we show an SGX-based DRAM attack abusing the SGX protection features to conceal both high cache miss and high bank locality features. Second, we present a one-row DRAM attack, a new type of DRAM attack, to further relax certain attack conditions.

*1) SGX-based attacks:* SGX runs in a secure, trusted environment with hardware isolation, where even the OS cannot access its memory. According to Intel, SGX enclave activity is not visible in the thread-specific performance counters [4] [16]. Therefore, SGX-based attacks can conceal both high cache miss and high bank locality features, and hence defeat existing detection approaches.

Successful SGX-based DRAM attacks require two primitives: a high-resolution timer to distinguish row buffer hits and misses, and a method to generate a set of addresses that map to target DRAM rows. For the high-resolution timer, on SGX2 [13], *rdtsc* is available within enclaves. On SGX1 [13], [4] demonstrated that accurate timing can be obtained by using counting threads, and [17] mirrored *rdtsc* into the enclave. Our experiments with both approaches show that we can use either technique to obtain sufficiently accurate timing inside the enclaves. Therefore, we focus on the method of generating the address sets. As shown in Fig. 3, enclaves occupy a piece of reserved DRAM, EPC, that does not share a bank with other DRAM. To launch DRAM attacks on all DRAM banks, we generate two address sets: one *application address set* mapping to all no-EPC banks using a traditional technique [1] and one *enclave address set* mapping to all EPC banks using the methods in [4]. According to [13], an enclave can access addresses out of EPC but in the same application, and we have verified that the access possesses the same hidden features as accesses to EPC addresses.

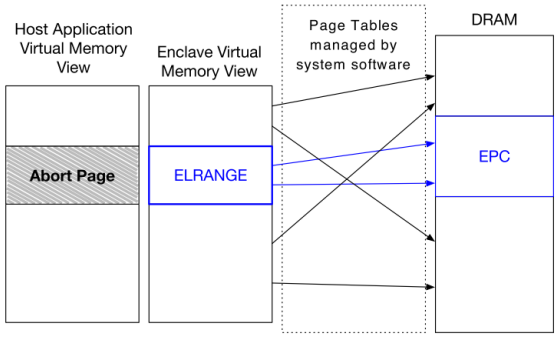



Fig. 3. An enclave's address map [13].

*2) One-row attacks:* As we have shown, existing DRAM side channel attacks need to allocate at least two memory blocks that map to the same DRAM bank, with one sharing the same DRAM row with a victim address and the other mapped to a different row on the same bank. To further relax the attack conditions, we introduce a new attack primitive, denoted as *one-row DRAM attack*. Modern systems employ sophisticated memory controller policies, preemptively closing rows earlier than necessary to optimize performance [18]–[20]. We conjecture that this policy creates a previously unknown DRAM effect, which we exploit with a one-row DRAM attack.

As shown in Fig. 4, with a one-row DRAM attack, the attacker simply runs a Flush+Reload loop on a single memory address (a). This will continuously reopen the same DRAM row, whenever the memory controller closes the row. If the victim address (v) is not accessed, the attacker always has a row buffer miss with a longer access time, while the victim address (v) is accessed, the attacker will obtain a row buffer hit with a shorter access time. Therefore, a one-row DRAM side channel attack is successful. In addition, a one-row DRAM covert channel attack can also be successful using a similar method.

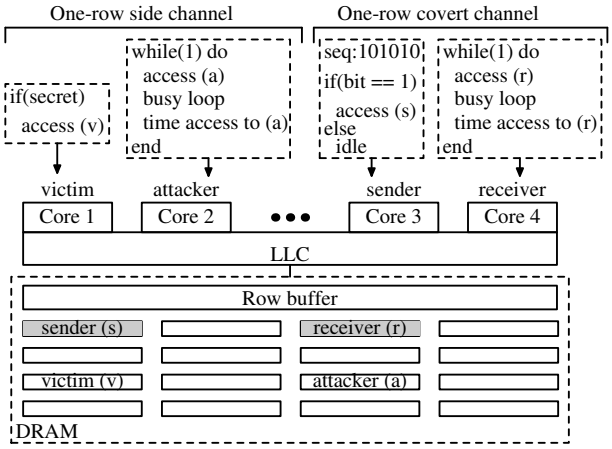



Fig. 4. A one-row DRAM attacker only needs one address sharing the same DRAM row with a victim (sender) address

*3) Attack Implementations:* To fully demonstrate the attack effect, we implemented four types of DRAM attacks: a traditional multi-row non-SGX attack, a one-row non-SGX attack, a multi-row SGX-based attack and a one-row SGX-based attack. For each type of attack, we implemented DRAM side and covert channel attacks. We implemented the attacks on an Ubuntu-based server with two Intel core i7-9700k processor (Coffee Lake) with an 8 GB DDR4 DRAM modules.

Table I shows the performances of the four types of DRAM attacks. We use the attack accuracy (i.e., successfully transmitting the secret value) of 1000 accesses to a victim address to measure side channel attacks, and use the channel capacity to measure covert channel attacks. As shown in the table, our SGX-based attack achieves similar attack performances to existing non-SGX attacks, but is harder to be detected. Although the attack performance of the one-row attack is slightly worse than traditional multi-row attacks, it still presents a severe threat.

TABLE I
DRAM ATTACK EFFECT OF FOUR TYPES OF DRAM ATTACKS.

| **Attack technique** | **Side channel** (Accuracy) | **Covert channel** (Capacity) | **Stealthy** |
|---|---|---|---|
| Multi-row without SGX | 81% | 40 kbps | × |
| One-row without SGX | 72% | 12 kbps | × |
| Multi-row with SGX | 79% | 40 kbps | ✓ |
| One-row with SGX | 71% | 13 kbps | ✓ |

## IV. OUR SOLUTION DRAMD

In this section, we present a software-based solution DRAMD, to detect both traditional DRAM side and covert channel attacks and the advanced SGX-based DRAM attacks.

DRAMD monitors several hardware performance counters, learns different applications' patterns, and builds a Convolutional Neural Network (CNN) to detect stealthy channels in SGX. Although the hardware performance counters cannot reveal anything inside SGX, our solution still works since SGX relies on the OS and unwillingly leaks information.

In this section, we first discuss how the information in SGX leaks to the OS, then presents how DRAMD monitors applications for detecting attacks at runtime, and then presents the detail design of our neural network based detection method used to detect SGX-based DRAM attacks.

### A. Information Leak in SGX

Although the OS and the hardware (including performance counters) cannot directly reveal information of SGX, we figure out SGX still indirectly leaks information to the OS.

*1) SGX process scheduling:* In modern multitasking processors, multiple processes often need to share time slices of the same CPU core, which requires the OS to schedule these processes [21]. Linux process scheduling is based on the time sharing and process preemption technique, in which the CPU time is divided into slices, one for each runnable process. If a currently running process is not terminated when its time slice expires, or a process with a higher dynamic priority is ready, a process switch may occur. Linux process scheduling relies on timer interrupts without the help of processes being scheduled. No additional code need to be inserted into the programs/processes to perform process scheduling [22], [23].

However, SGX-enabled processors use the same process scheduling policy as normal processors, and the SGX process is still scheduled by the OS [13]. This is the root cause of SGX information leakage discussed in this paper.

*2) Leaking via Process Scheduling:* During process scheduling, the core running the SGX processes needs to interact with the OS with some necessary information, which eventually reveals actions inside the SGX. More specifically, the running core needs to periodically report the status information necessary for process scheduling to the OS. These status information are actually depending on the core's operations (i.e., the SGX application code). So, we could retrieve certain information by analyzing the status information. Specifically, when sending and receiving process scheduling information to and from the OS, CPU cores running different SGX applications have different visible interaction patterns, which can be used to identify actions inside the SGX.

*3) Example:* As shown in Fig. 5, three groups of operations within SGX, e.g., *no memory access*, *memory access with cache hit* and *memory access with cache miss*, cause different times of invocations to the system function $update\_curr()$, making them detectable. In the Linux scheduling strategy, the periodic scheduler is started periodically during the execution of a process, and invokes $update\_curr()$ function to update

some related data. Since memory access with cache hit is faster than memory access with cache miss, the former will release the CPU more frequently than the latter, causing more invocations to $update\_curr()$. In this way, we could infer certain information of the SGX application internals.

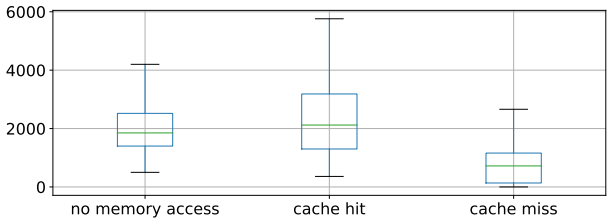



Fig. 5. The number of accesses to the $update\_curr()$ function in 30 seconds for three SGX with different operations: no memory access, memory access with cache hit and memory access with cache miss.

### B. Overview

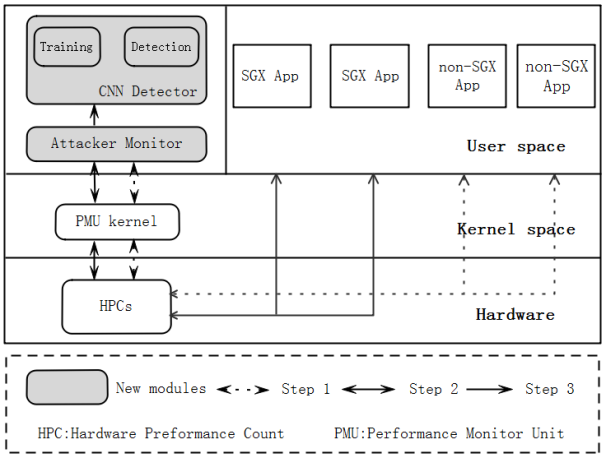



Fig. 6. Architecture overview of DRAMD.

Fig. 6 shows the architecture of DRAMD and the workflow for detecting DRAM attacks. For completeness, we designed DRAMD to detect both traditional non-SGX DRAM attacks and SGX-based DRAM attacks. The only features used by DRAMD are hardware event values read from the hardware performance counters available in commercial processors.

We implement DRAMD as a group of services in the host OS kernel. DRAMD consists of two modules, each running on a dedicated core. The *Attacker Monitor* is responsible for collecting cache activities of non-SGX apps, therein using the high cache miss feature to detect traditional DRAM attacks, and collecting another 4 runtime events of SGX applications: cache misses (the number of cache misses), branches (the number of branches executed), instructions (the number of instructions executed) and branch misses (the number of branch prediction failures). These 4 runtime events will be fed to the *CNN Detector* to learn and detect SGX-based DRAM attacks using our neural network based classifier.

### C. Performance Events Monitoring

At runtime, DRAMD keeps monitoring both non-SGX and SGX applications using hardware performance counters. Specifically, DRAMD monitors all non-SGX applications running on a server to detect non-SGX DRAM attacks using the high cache miss feature. Meanwhile, DRAMD monitors all SGX applications running on the server to collect their interaction patterns with the OS, and uses a neural network based method to detect SGX-based DRAM attacks.

Fig. 6 shows the workflow of DRAMD. Details of each step are described as follows:

**Step 1: Monitoring cache activities of non-SGX apps.** This step occurs at runtime. The *Attacker Monitor* exploits the performance counters to monitor all non-SGX apps simultaneously. The *Attacker Monitor* determines whether it is a non-SGX app by checking whether the CPU is in enclave mode. One challenge is that there are no enough performance counters available on the servers to monitor all non-SGX applications, since most Intel and AMD processors support up to six counters, and the number of counters does not scale with the number of cores. Thus, when there are many non-SGX apps on the server, the *Attacker Monitor* cannot monitor them concurrently.

To solve this problem, we use a time-domain multiplexing method: the *Attacker Monitor* identifies active CPU that runs non-SGX apps and then measures each of them in turn. If one CPU has a high cache miss, the *Attacker Monitor* will flag an alarm.

**Step 2: Monitoring 4 runtime events of SGX apps.** This step occurs concurrently with Step 1. The *Attacker Monitor* exploits performance counters to monitor all SGX apps simultaneously. It periodically (e.g., every 1 ms) records the 4 event counts (cache misses, branches executed, instructions executed and branch misses) which will be fed to the *CNN Detector*.

The *Attacker Monitor* still determines whether it is an SGX app by checking whether the CPU is in enclave mode. In addition, we also use the time-domain multiplexing method in Step 1 to solve the challenge of insufficiency of performance counters available on the server to monitor many SGX apps.

**Step 3: Detecting SGX-based DRAM attacks.** This step occurs at runtime. The *CNN Detector* periodically receives the 4 event counts and regards each 1000 events as one sample (the effect of the number of events in a single sample on the detection effect will be evaluated in the evaluation section). It continues detecting the most recent data sample, and will flag an alarm if an attack is detected.

### D. Neural Network based Detection

DRAMD uses a neural network classifier to detect SGX-based attacks. Features and classifiers directly determine the detection effect of neural networks, hence we need to select a proper hardware performance feature and design a suitable neural network classifier.

*1) Feature Selection:* We choose 4 hardware events (cache misses, branches executed, instructions executed and branch misses) as the final features. This is because the interaction patterns with OS, used for attack detection are characterized

by the cache miss count, the branch executed count, the instruction executed count and the branch miss count, measured by the performance counters.

As shown in the previous subsection, there are hundreds of functions in the OS that are responsible for process scheduling, and SGX actions can be identified by the number distribution of *OS process scheduling functions* invoked when the SGX application is being scheduled. However, we have no way to record the number of OS process scheduling functions invoked by a certain SGX. Fortunately, DRAMD only needs to know the relative number of invokes to these functions, which can be inferred by the number of cache misses, branches executed, instructions executed or branch misses. These events will be generated when these OS functions are executing, and be recorded in the hardware performance counters. To improve the detection accuracy, we combined all these 4 hardware events for the neural network to make decisions.

*2) CNN Classifier:* We choose CNN as the final classifier. CNN is a branch of machine learning, which has been proved very effective for signal recognition tasks such as speech transcription, image segmentation, image classification, and many others [24]. CNNs are good at capturing high-level concepts that are easy for humans to agree on but hard to express formally. In our case, we use CNNs to capture interaction patterns of SGX process and OS for a given SGX action, even in the presence of some variations among these actions.

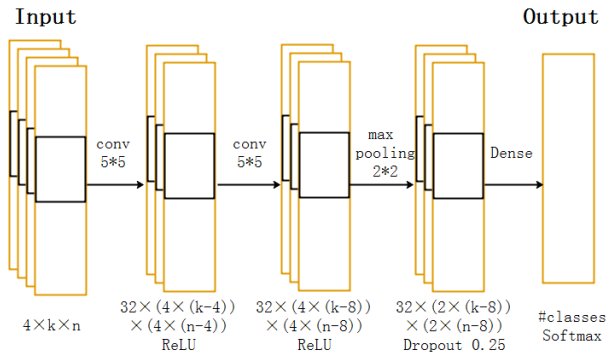



Fig. 7. Our CNN architecture. The input has 4 layers denoting the 4 hardware events (cache misses, branches executed, instructions executed and branch misses). $k \times n$ denotes the number of OS process scheduling functions, which is 864 in our CNN classifier.

We use CNNs with two convolution layers, one max pooling layer, and one dense layer (Fig. 7). We train them using an Adam [25] optimizer on batches of 64 samples, with the categorical cross-entropy as the error function. The classifier is constructed using TensorFlow with the Keras front end. The CNN input is the number distribution of 1000 hardware events in OS process scheduling functions, and each 1000 events is regarded as one sample. The CNN output indicates whether a sample is a DRAM attack. We use supervised training on a corpus that consists of DRAM attacks and some normal actions, labeled with their correct class. The selection of normal actions and the generation of labeled training data will be discussed in Sec. V.

## V. Evaluation

In this section, we evaluate the security and performance of DRAMD to validate its design and implementation.

### A. Setup

We use a Coffee Lake server with a 3.6-GHz Intel Core i7-9700k CPU. The processor contains 8 physical cores (hyperthreading is not supported) sharing a 12-MB LLC. Each core has a 32-KB L1 data and instruction cache, and a 256-KB L2 unified cache. The server is equipped with two 8 GB DDR4-2400 DIMMs. We used the latest official Intel SGX lib (Version 2.6), driver (Version 2.5) and SDK (Version 2.6) [26].

We implemented DRAMD as a kernel service with ∼2000 lines of code in the Linux kernel 4.15.0-39-generic running Ubuntu 16.04.3 TLS.

### B. Security Evaluation

*1) Classification Accuacy of SGX Application Behaviors:* We first show that our CNN classifier can accurately identify different actions in SGX. As shown in Sec. IV, we used 4 hardware events (cache misses, branches executed, instructions executed and branch misses) as the feature. We use the SGX-based DRAM attack (action with high cache miss), including multi-row attack and one-row attack, and 7 common normal actions as classification targets: normal memory access (action with high cache hit), RSA, ECC, AES, DES, MD5 and SHA512 actions.

**Dataset.** To collect the dataset, an SGX occupying a dedicated core performs a target action and the DRAMD occupying another dedicated core records the 4 hardware events. Since our classifier essentially takes advantage of the statistics of these events, a sample should contain enough events to cover most OS process scheduling functions. We explored the effect of changing the number of events in a single sample on the detection accuracy. We chose three types of samples with different numbers of events, i.e., 500, 1000 and 2000 events, and generated all datasets three times.

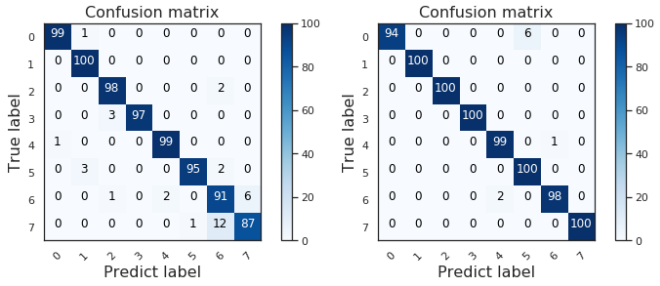
*a) Training dataset:* We first generated training data for each action in SGX. An SGX continues running an action until the DRAMD collects enough samples. We collected 1000 training samples for each target action in SGX.

*b) Test dataset:* We used the same method to collect the test data, and we collected 500 test samples for each target action in SGX.

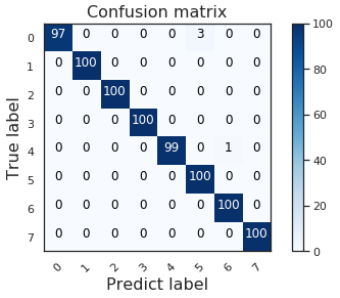
**Result.** We consider the identification of an action in SGX as a multi-classification and measure its confusion matrix. Fig. 8 shows the results of 8 trained classes under three types of samples with different numbers of events. From this figure, we can see that 1000 events and 2000 events give better accuracy than 500 events: our classifier can achieve more than 94% accuracy for all classes, and there are very few false positives. For 500 events, normal memory access, RSA, ECC and AES actions can be identified with high accuracy, whereas other actions cannot be differentiated from each other with reasonable false positive and false negative rates. This is

because the 500 events cannot capture sufficient information to cover most OS process scheduling functions.

The optimal number of events in a sample depends on how an SGX core invokes OS process scheduling functions. If the sampling time for a single sample (the time required to collect a certain number of events) is much longer than the time for invoking most functions, the sample will contain more data points, thus yielding more accurate results. In our experiments, a sample with 1000 events (taking approximately 5 s to collect), the default setting in DRAMD, can cover most OS process scheduling functions, providing good results for all eight classes.




(a) The sample of 500 events (b) The sample of 1000 events



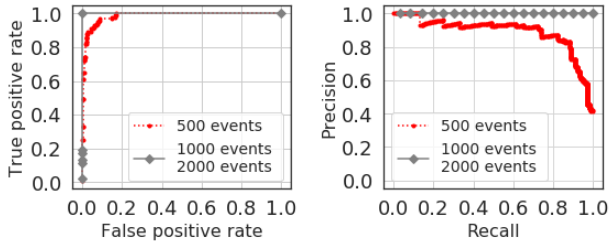

(c) The sample of 2000 events

Fig. 8. Confusion matrix of 8 target actions. 0:DRAM attack, 1:normal memory access, 2:RSA, 3:ECC, 4:AES, 5:DES, 6:MD5, 7:SHA512.

*2) SGX-based DRAM Attack Detection Accuracy:* Since the detection accuracy of traditional non-SGX DRAM attacks has been demonstrated in [12], we only measure the detection accuracy of SGX-based DRAM attacks using the 4 hardware events as the feature.

We considered the detection of an SGX-based DRAM attack as a binary classification, and we measured its true positive rate and false positive rate in addition to the precision and recall. We used the same 8 target actions in SGX as in the previous subsection. DRAMD first generates training data for the SGX-based DRAM attack and normal actions (the other 7 actions). We use the same method to generate a training dataset to train our classifier except that here we regard actions other than DRAM attacks as the "normal action" class. In the detection phase, an SGX occupying a dedicated core runs all types of DRAM attacks (side and covert channels, and multi-row attacks and one-row attacks). 4 SGX occupying 4 other dedicated cores randomly run normal actions, including untrained actions. Meanwhile DRAMD occupying another dedicated core detects the attacks. We repeated the experiment 100 times and measured the number of true positives and false positives under different thresholds. We plot the *receiver operating characteristic* (ROC) and *precision-recall* (PR) curves to show the relations between the true positive rate and false positive rate, and those between the precision and recall.

We also tested the three types of samples with different numbers of events. Fig. 9 shows the results of detecting SGX-based DRAM attacks under the three types of samples. From this figure, we can see that 1000 events and 2000 events give better accuracy than 500 events: DRAMD can achieve a true positive rate close to 100 % with a zero false positive rate. For 500 events, the attacks cannot be differentiated from normal actions with reasonable false positive and false negative rates. This further validates the conclusions in our previous section.




(a) ROC (b) PR

Fig. 9. PR and ROC curves of SGX-based DRAM attack detection.

### C. Performance Evaluation

Our intent was to test the performance penalty to the host due to DRAMD with the default settings in a high-load system. We used 20 SPEC CPU 2017 benchmarks (the full SPEC 2017) [27] and 13 PARSEC benchmarks (the full PARSEC) [28] for a performance evaluation. We launched 8 dockers on the machine, which acted as real applications, each using one of eight cloud applications from CloudSuite [29] (data analytics, data caching, data serving, graph analytics, in-memory analytics, media streaming, web searching and web serving). We show the normalized run time for each benchmark in Fig. 10 (the results averaged over 10 runs). Note that most benchmarks are unaffected (with results of approximately 1). Some benchmarks are affected, therein becoming slower (with results greater than 1). In summary, the results suggest that DRAMD has little impact on the performance of benign applications; and even in the worst case, the performance overhead is less than 5%.

## VI. Related Work

### A. Hardware Side and Covert Channels

Attacks exploiting hardware sharing can be grouped into two categories. In side channel attacks, an attacker spies on a victim and extracts sensitive information such as cryptographic keys [30], [31]. In covert channel attacks, the sender and receiver bypass conventional security mechanisms, allowing unmonitored communications between two unconcerned entities [32], [33].

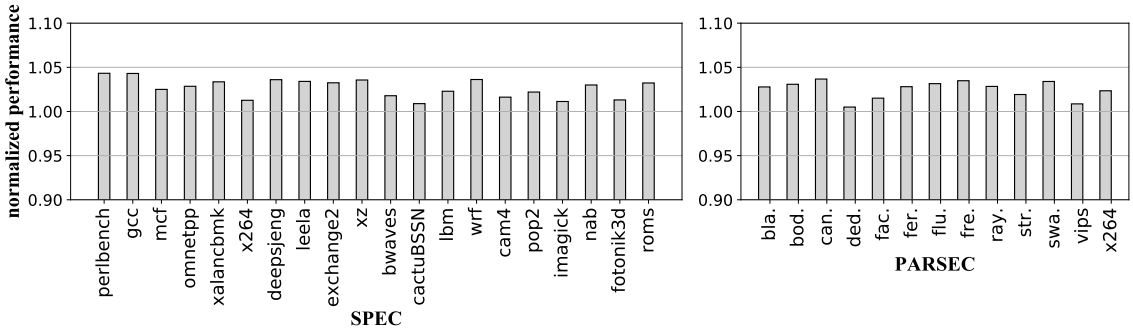



Fig. 10. Performance of different benchmarks. The x-axis shows the benchmarks, and the y-axis shows the normalized run time of each benchmark, which is the ratio of the time during which DRAMD is enabled to the time during which DRAMD is disabled.

*1) DRAM attacks:* [1] found a new attack vector that targets the row buffer in DRAM modules. While accesses to this buffer are fast, accesses to other memory locations in DRAM are much slower. This timing difference can be exploited to obtain fine-grained information across virtual machine boundaries. [4] exploits the DRAM row buffer timing differences to recover an eviction cache set from a virtual address without relying on large pages in the SGX enclave. [17] implements the attack in the SGX enclave to extract sensitive information in a cross-enclave environment.

*2) Cache attacks:* Side and covert channels using the CPU cache exploit the fact that cache hits are faster than cache misses. The methods Prime+Probe [3], [34] and Flush+Reload [35] (along with its variant, Flush+Flush [36]) have been presented to build either side or covert channels. Attacks targeting the last-level cache are cross-core attacks but require the sender and receiver to operate on the same physical CPU. [35] presented the first LLC side channel attacks using Flush+Reload. [3] introduced an effective implementation of the Prime+Probe-based side channel attack against the LLC. [36] demonstrated a stealthy LLC attack using Flush+Flush.

*3) SGX attacks:* Various micro-architectural side and covert channel attacks have been demonstrated on SGX, including CPU cache attacks [37], [38], BTB attacks [39], page-table attacks [40], and cache-DRAM attacks [17].

### B. Countermeasure

This paper is the first work dedicated to detect DRAM channels. Only a few straightforward methods for detecting non-SGX DRAM attacks have been presented in previous work and only as a minor aspect. Most existing countermeasures focus only on cache channels and SGX channels.

*1) Cache channel defenses:* Existing cache countermeasures can be categorized into hardware- and software-based solutions. Hardware-based solutions focus on new cache designs, such as partitioned caches [5], randomized/remapping caches [41], [42], and line-locking caches [6], [7]. Software-based solutions can be divided into three categories: detection countermeasures [8], [9], neutralization countermeasures [10], and elimination countermeasures [43], [44].

*2) SGX channel defenses:* Most known defenses are designed specifically toward page-fault side channel attacks. [45] proposed a compiler-based approach to transform cryptographic programs to hide page access patterns that may leak information. [46] proposed T-SGX, which exploits Intel's transactional synchronization extensions (TSX), to prevent page faults from revealing the faulting address. [47] proposed a secure enclave architecture that is similar to SGX but that is resilient to both page-fault and cache side channel attacks.

## VII. Conclusion

DRAM side and covert channel attacks pose severe threats to system security. In this paper, our findings are two-fold. First, we demonstrate that existing detection techniques based on high cache miss and high bank locality features cannot detect SGX-based DRAM attacks. We also present one-row DRAM attack, a new type of DRAM attack, to relax certain attack conditions. Second, we design DRAMD, a software system to detect both traditional non-SGX DRAM attacks and SGX-based DRAM attacks. DRAMD leverages the existing hardware performance counters to capture the high cache miss features of non-SGX attacks, and to track OS functions that are responsible for SGX process scheduling to detect DRAM attack activity concealing in SGX. DRAMD is designed as a lightweight service in the OS kernel and does not require any new hardware, OS, or application modifications. The feasibility of DRAMD is validated by our implementation on the Ubuntu 16.04 OS kernel. Our evaluation shows that DRAMD can effectively detect DRAM attacks while simultaneously introducing little overhead to benign applications.

## VIII. Acknowledgement

This work was supported in part by National Natural Science Foundation of China under Grant 61772308, 61972224 and U1736209, and BNRist Network and Software Security Research Program under Grant BNR2019TD01004 and BNR2019RC01009.

## References

[1] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA}: Exploiting {DRAM} addressing for cross-cpu attacks," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 565–581.

[2] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches." in *USENIX Security Symposium*, 2015, pp. 897–912.

[3] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 605–622.

[4] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using sgx to conceal cache attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 3–24.

[5] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "Secdcp: Secure dynamic cache partitioning for efficient timing channel protection," in *Design Automation Conference*, 2016, pp. 1–6.

[6] Y. Chen, M. Khandaker, and Z. Wang, "Secure in-cache execution," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2017, pp. 381–402.

[7] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (sharp): Defending against cache-based side channel attacks," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 347–360.

[8] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 118–140.

[9] M. Yan, Y. Shalabi, and J. Torrellas, "Replayconfusion: detecting cache-based covert channel attacks using record and replay," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 39.

[10] A. O. F. Atya, Z. Qian, S. V. Krishnamurthy, T. La Porta, P. McDaniel, and L. Marvel, "Malicious co-residency on the cloud: Attacks and defense," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[11] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 19–35.

[12] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

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

[14] C. E. A. Shannon, "A mathematical theory of communication. at&t tech j," *Acm Sigmobile Mobile Computing & Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[15] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer, "Hello from the other side: Ssh over robust cache covert channels in the cloud," *NDSS, San Diego, CA, US*, 2017.

[16] IntelCorporation, "Intel sgx: Debug, production, pre-release what's the difference?" https://software.intel.com/en-us/blogs/2016/01/07/intelsgx-debug-production-prelease-whats-the-difference.

[17] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2421–2434.

[18] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 2011, pp. 31–40.

[19] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 24–35.

[20] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.

[21] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting task migration in multi-processor systems-on-chip: a feasibility study," in *Proceedings of the Design Automation & Test in Europe Conference*, vol. 1. IEEE, 2006, pp. 1–6.

[22] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[23] R. Love, *Linux Kernel Development: Linux Kernel Development _p3*. Pearson Education, 2010.

[24] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1357–1374.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] IntelCorporation, "Intel-software-guard-extensions/downloads," https://01.org/zh/intel-software-guard-extensions/downloads?langredirect=1.

[27] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: ACM, 2018, pp. 41–42. [Online]. Available: http://doi.acm.org/10.1145/3185768.3185771

[28] C. Bienia, "Benchmarking modern multiprocessors," *Dissertations & Theses - Gradworks*, 2011.

[29] EPFL:Cloudsuite, http://cloudsuite.ch/.

[30] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 490–505.

[31] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 305–316.

[32] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud." in *USENIX Security symposium*, 2012, pp. 159–173.

[33] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," *Computer Science*, 2015.

[34] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S $ a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 591–604.

[35] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *Usenix Conference on Security Symposium*, 2014, pp. 719–732.

[36] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.

[37] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2017, p. 2.

[38] M. Hähnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, 2017, pp. 299–312.

[39] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 557–574.

[40] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1041–1056.

[41] F. Liu, H. Wu, K. Mai, and R. B. Lee, "Newcache: Secure cache architecture thwarting cache side-channel attacks," *IEEE Micro*, vol. 36, no. 5, pp. 8–16, 2016.

[42] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *Design Automation Conference*, 2017, p. 7.

[43] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *IEEE International Symposium on High PERFORMANCE Computer Architecture*, 2016, pp. 406–418.

[44] X. Meng, L. Thi, P. Xuan, H. Y. Choi, and I. Lee, "vcat: Dynamic cache management using cat virtualization," in *Real-time & Embedded Technology & Applications Symposium*, 2017, pp. 211–222.

[45] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 317–328.

[46] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.

[47] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 857–874.