Meet the Sherlock Holmes' of Side Channel Leakage: A Survey of Cache SCA Detection Techniques

Ayaz Akram, Maria Mushtaq, Muhammad Khurram Bhatti, Vianney Lapotre, Guy Gogniat

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 20, 2020, accepted February 26, 2020, date of publication March 13, 2020, date of current version April 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2980522

Meet the Sherlock Holmes' of Side Channel Leakage: A Survey of Cache SCA Detection Techniques

AYAZ AKRAM[1], MARIA MUSHTAQ[2], MUHAMMAD KHURRAM BHATTI[3], VIANNEY LAPOTRE[4], AND GUY GOGNIAT[4]

[1]Department of Computer Science, University of California, Davis, Davis, CA 95616, USA
[2]LIRMM, CNRS, Université de Montpellier, 34095 Montpellier, France
[3]Department of Electrical Engineering, Information Technology University, Lahore 54890, Pakistan
[4]Lab–STICC, University of South Brittany, 56100 Lorient, France

Corresponding author: Ayaz Akram (yazakram@ucdavis.edu)

This work was supported in part by the PHC PERIDOT Project e-health SECURE under Grant 3-6/HEC/R&D/PERIDOT/2017.

ABSTRACT Cache Side Channel Attacks (SCAs) have gained a lot of attention in the recent past. Since, these attacks exploit the caching hardware vulnerabilities, they are fast and dangerous. Detection of cache side channel attacks is an important step towards mitigating against such hostile entities. Researchers have already proposed different techniques to detect cache side channel attacks. This paper provides a detailed survey of literature related to the state-of-the-art detection techniques for cache based side channel attacks. We identify a set of important characteristics that can be used to characterize a CSCA (cache side channel attack) detection technique. We use the identified features to compare and contrast the most important detection techniques and provide the important observations. We also identify some of the challenges that the research community will have to resolve in future to improve the efficiency of cache side channel detection techniques. To the best of our knowledge, this is the first work to do such a study. We believe that this paper will prove useful to researchers in the area of systems security.

INDEX TERMS Cache-based side channel attacks (SCAs), cryptography, survey, detection, machine learning, anomaly & signature detection.

I. INTRODUCTION

Information security is becoming a major concern with each passing day as innovative and smart security attacks keep on appearing. With the emergence of new fields like IoT, Blockchain, Cloud-Computing and Cyber-Physical Systems (CPS), the amount of produced digital data has increased exponentially over the past few years. Roughly 2.5 quintillion bytes of data is produced each single day according to IBM Big Data Research. Since it is expensive to process such big amounts of data on end-user devices, any required processing on this data is usually done in centralized computing environments (i.e. Cloud Systems). Cloud Computing platforms have been continuously evolving with their increasing use. Current cloud computing facilities are provided in various forms like Software-as-a-Service (SaaS), Platform as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [1]. Such services extensively use virtualization technology to provide isolated processing capability to users without any interference. Multiple Virtual Machines (VMs) are executed on the shared hardware resources which create a possibility of security breaches. Malicious VMs that share hardware resources (co-reside) with other VMs (referred as victim VMs) can obtain their information [2]–[4] and perform Side Channel Attacks (SCAs) on victim VMs [5], [6]. Significant amount of research has been performed in the field of cryptography leading to the development of different crypto-algorithms like ECC, AES, RSA, and ElGamal etc. Theoretically, these algorithms are very hard to break and require enormous computing power. For instance, for a 128-bit AES key, it would take 5.4×10^{18} years to crack the AES using a computer capable of performing 10^6 decryption operations per μs [7]. However, many research works have shown that crypto-systems, such as AES, can be compromised due to the vulnerabilities of the hardware on which they run. Side Channel Attacks (SCAs) do not target the algorithm of crypto-systems itself. Rather, they target the underlying implementation of systems on which these crypto-systems execute [8] as shown

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Ayoub Khan.

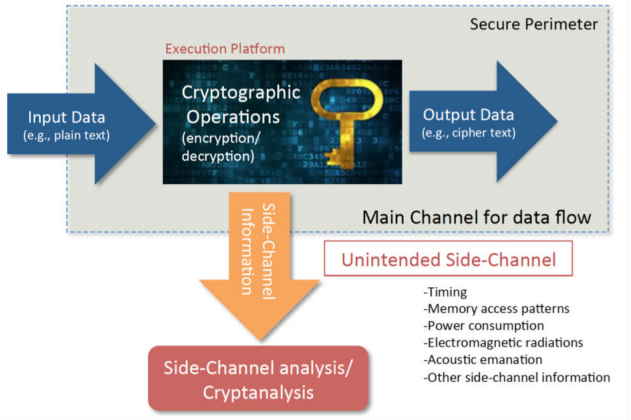


FIGURE 1. Unintended Side-Channel information leakage.

in Figure 1. SCAs can use a variety of physical parameters, e.g., power consumption [9], EM (electro-magnetic) radiation [10], memory accesses or fault occurrence [5], [11]–[18] to extract secret keys/information. The baseline idea here is that the SCAs can analyze the variations in the previously mentioned parameters during the execution of crypto-systems on a particular hardware and can determine the secret information used by crypto-systems based on the observed parameters.

In modern processors, memory is used in a hierarchy (registers, caches, main memory (RAM) and storage) to improve performance of running processes by exploitation of spatial and temporal locality found in the accessed data. The level closest to the processor, i.e., registers, is fastest to access but limited in number. The next level of hierarchy is composed of caches that are used to hold part of the main memory, which is to be accessed frequently. Modern Intel processors consist of hierarchical caches up to three different levels (i.e., L1, L2, and L3) that vary in their sizes and response times. Last level of cache (L3) is usually shared among multiple cores in modern Intel processors. Cache side channel attack (CSCA) is a special type of SCA in which a malicious process deduces secret information of a victim process by observing its use of caching hardware [19]. In this paper, we focus on Cache Side Channel Attacks (CSCAs) in Intel' x86 Architecture based processors. Different cache properties of Intel's Architecture like inclusivity and flushing have also been used as vulnerabilities [5], [14], [20]. Inclusivity ensures that the cache blocks or lines present in higher level of cache would also be present in lower level caches. Flushing (eviction of any cache line with a particular address in all levels of cache) has been made possible with the help of instructions like *CLFLUSH* in Intel's x86 Instruction Set Architecture (ISA). CSCAs on other architectures like ARM have also been shown to be practical [21]–[23].

Contributions: This paper presents a survey of techniques that have been proposed to detect cache based side channel attacks. We classify these techniques into different categories based on their design characteristics. We provide a comparison of these techniques using a broad set of parameters and

TABLE 1. List of acronyms.

Name	Abbreviation
Side Channel Attack	SCA
Cache Side Channel Attack	CSCA
Evict and Time	E+T
Flush+Reload	F+R
Flush+Flush	F+F
Hardware Performance Counter	HPC
Machine Learning	ML
Last Level Cache	LLC
Virtual Machine	VM
Attacker Address Space	AAS
Victim Address Space	VAS

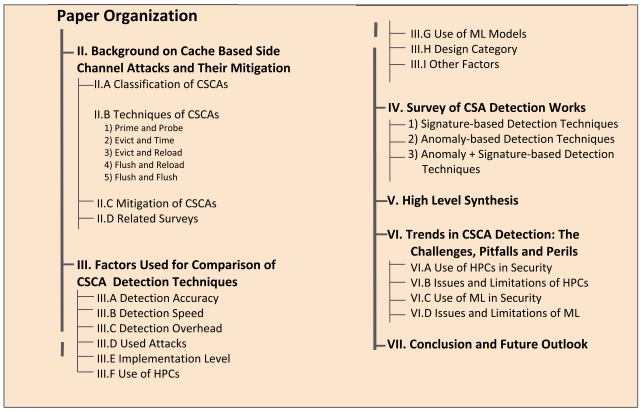


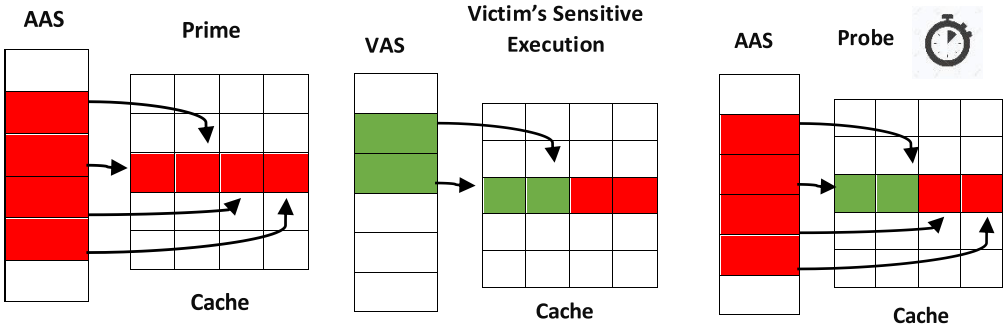
FIGURE 2. Paper organization.

enlist important findings, a high-level synthesis, and future possibilities based on our analysis of the surveyed techniques. A list of acronyms that will be used in this paper is shown in Table 1. The organization for the rest of this paper is shown in Figure 2. Section 2 discusses the background of CSCA and the most popular techniques that have been proposed to perform and mitigate such attacks. Section 3 defines and explains the most important set of parameters that can be used to compare different CSCA detection mechanisms. Section 4 presents a detailed survey of the CSCA detection techniques found in literature. Section 5 provides a summary of the most significant findings of this study and Section 6 concludes this paper.

Scope: Many countermeasures and mitigation techniques have been proposed against CSCAs in literature. However, we limit the scope of this work to only CSCA detection approaches. CSCAs are possible on different attack targets. However, this work only considers CSCA attacks against crypto systems. Moreover, only CSCAs in x86 architecture are considered, while they have shown to be possible on other architectures (like ARM) as well.

II. BACKGROUND ON CACHE BASED SIDE CHANNEL ATTACKS AND THEIR MITIGATION

In this section, we will discuss the classification of cache side channel attacks, different techniques that have been



(a) Prime phase (b) Execution of victim (c) Probe Phase

FIGURE 3. Working principle of Prime+Probe (AAS: Attacker Addr. Space, VAS: Victim Addr. Space).

proposed to perform such attacks and mitigation solutions that have been presented against these cache side channel attacks. We will also discuss any works that are related to our paper. Readers who already have a background knowledge of cache side channel attacks can skip this section.

A. CLASSIFICATION OF CACHE SIDE CHANNEL ATTACKS

Different classifications of CSCAs based on different factors can be found in literature [24], [25]. The inherent feature that most of the cache side channel attacks exploit is the cache timing and access patterns. In order to make an attack possible, usually the attacker needs to identify the victim's memory addresses by observing the shared cache with the victim. Further, an attacker would also have to determine if a particular memory access by the victim results into a hit or a miss. Mostly attackers observe the victim's memory accesses indirectly rather than directly. Therefore, cache side channel attacks can be classified into Access Based Attacks (e.g. Prime+Probe, Flush+Reload) and Timing Based Attacks (Evict-and-Time, Cache collision) [24]. In Access-Based attacks, after the interference of attacker with the victim, the attacker observes the time that each of its memory access takes. It is assumed that the attacker will have logical access to a shared cache used by the victim process. In Timing Based Attacks unlike Access-Based attacks, attacker measures the execution time of a security critical operation performed by the victim rather than measuring the time of its own memory accesses [24].

Another classification of CSCAs is done by Page et al [25], according to which CSCAs are categorized into two types; Time driven and Trace attacks [25]. These attack types are based on the type of information leaked by the attack. Time-driven attacks commonly named as timing attacks depend on quantifiable execution time which relays information of secret cryptographic operations. This information is useful during the encryption process where number of cache hits and misses relay variable timing information. Timing difference of cache hits and misses allows the attacker to know the interesting addresses where victim is operating. Once this information is revealed, attacker is able to retrieve entire secret key. Thus, such timing attacks are capable of measuring entire execution time efficiently. Time-driven attacks are further categorized into active time-driven cache attacks and passive time-driven cache attacks. In passive time-driven attacks, attacker has no direct influence on victim's machine and attacker is unable to probe the timing information directly [26], [27]. In active time-driven attacks, the attacker actively influences the victim's machine because it can execute directly on victim's machine. Therefore, attacker is well informed about timing information of victim by which it can manipulate a lot of information [26], [28], [29]. Trace-driven attacks manipulate the trace of victim's accesses [25], [30]–[33]. This type of attack focuses to access the cache line which is frequently used by victim.

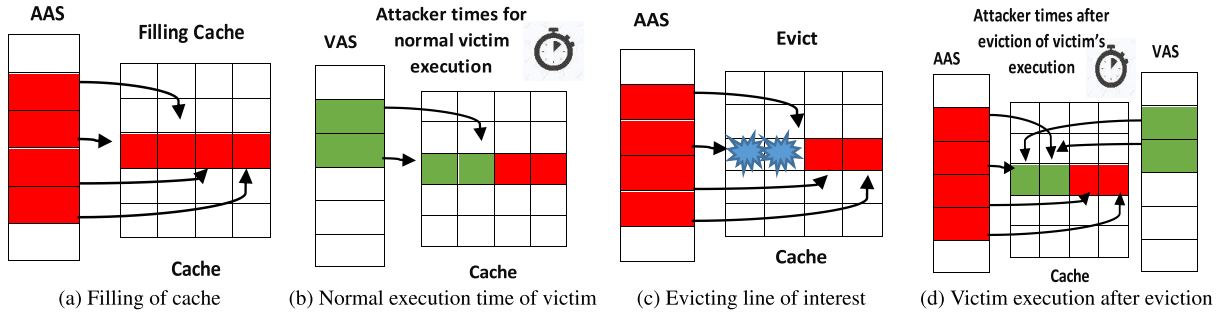
B. TECHNIQUES OF CACHE SIDE CHANNEL ATTACKS

This sub-section discusses most of the famous techniques that have been proposed to perform CSCAs.

1) PRIME AND PROBE TECHNIQUE

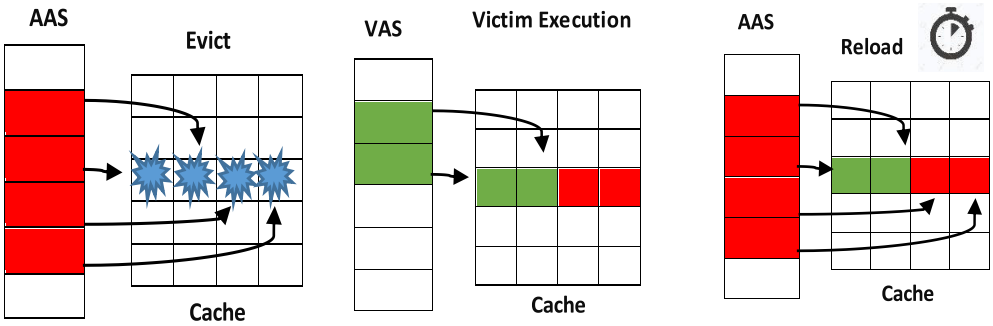
Prime+Probe attacks belong to the category of Trace Driven attacks and are common to exploit last level shared caches across multiple cores. Many CSCAs have made use of this technique: [6], [12], [20], [22], [34]–[39]. A prime+probe attack works in two phases (prime phase and probe phase) as show in Figure 3. During the prime phase, attacker fills the shared cache with its own data as shown in Figure 3a. Attacker then goes to an idle mode and waits for victim to execute which will use few of the cache sets primed by the attacker as shown in Figure 3b. During the probe phase (shown in Figure 3c), attacker tries to load the data from the cache that it had primed (filled) earlier. If any of the cache sets have been overwritten by the victim process, it will take longer for attacker to access them (as they will have to be brought from main memory due to cache replacement). This way, attacker can get to know if the victim has used particular addresses and can extract secret data from this information.

This technique has been used at different cache levels like L1-data (L1-D) cache [35], [36], L1-instruction (L1-I) cache [40] and Last Level Cache (LLC) [41].



(a) Filling of cache (b) Normal execution time of victim (c) Evicting line of interest (d) Victim execution after eviction

FIGURE 4. Working principle of Evict+Time.



(a) Eviction of cache (b) Victim's execution (c) Reload to measure time

FIGURE 5. Working principle of Evict+Reload.

2) EVICT AND TIME TECHNIQUE

This technique falls under the class of Time-Driven attacks. It works in a three-step process as shown in Figure 4. In the first step, as shown in Figure 4a attacker adds its own data in the cache. In the second step, attacker lets the victim process to execute normally, loads back its cache sets and notes the time of execution of the victim as shown in Figure 4b. In the third step, as shown in Figure 4c the attacker program removes a cache set (call it 'X') and lets victim execute. Then attacker analyses the variation in execution time of the victim process based on which it can be inferred if the line 'X' was accessed by the victim process or not (Figure 4d). Some of the implementations based on this technique of attack are: [12], [16], [35], [42].

3) EVICT AND RELOAD TECHNIQUE

Evict+Reload (E+R) is a variation of Evict+Time (E+T) [15]. As shown in Figure 5, this technique works in two step. The first step is the eviction of cache sets shown in Figure 5a. The attacker then lets the victim run normally (Figure 5b). In the next step, the attacker reloads the data from cache and measures time of access to determine if it is a hit or a miss as shown in Figure 5c. Based on this information attacker can determine if the victim accessed particular address or not.

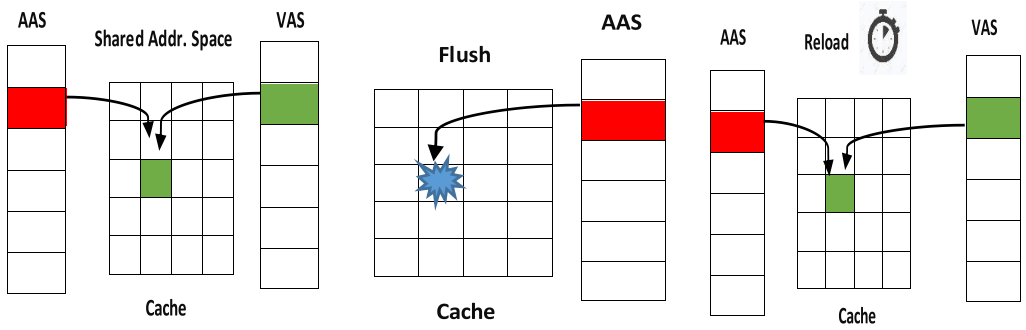
4) FLUSH AND RELOAD TECHNIQUE

Flush+Reload [5] shown in Figure 6, belonging to the class of Trace-Driven and Access Based attacks, relies on page sharing. There exist numerous cache attacks that used Flush+Reload technique: [11], [18], [20], [43]–[46]. In the first step of the attack (Flush), the attacker process evicts a shared cache set using available privileged instruction of *CLFLUSH*, shown in Fire 6b. Attacker lets the victim execute normally after flushing a shared cache line. In the next step of the attack (Reload), the attacker reloads the shared cache line and measures the loading time as shown in Figure 6c. The measured time indicates if the shared cache line was accessed by the victim or not.

5) FLUSH AND FLUSH TECHNIQUE

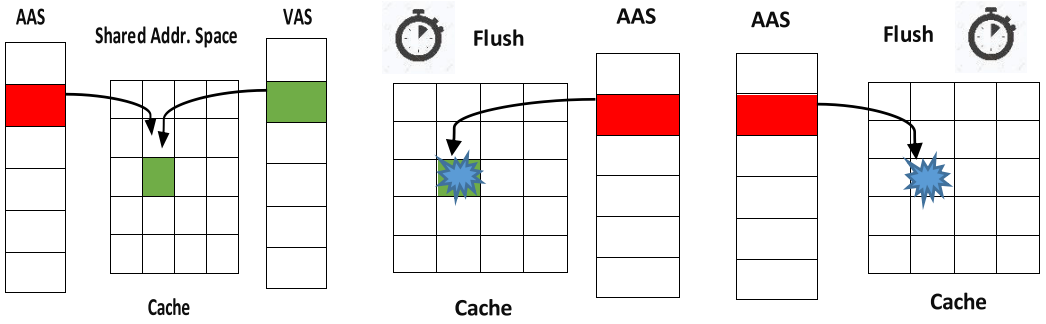
Flush+Flush [14], also belongs to the class of Trace-Driven attacks and replaces the Reload step of a Flush+ Reload attack with a Flush. This attack relies on measuring the execution time of *CLFLUSH* instruction as shown in Figure 7.

In the first phase of the attack, the attacker evicts/ flushes a particular cache line from the shared (with the victim) address space 7b. The attacker then lets the victim process execute normally which may or may not access the flushed cache line. In the second phase, the attacker flushes the same cache line and measures the time that the flushing operation takes as shown in Figure 7c. Using the measured time of the flushing (which would vary depending on the existence of target cache line in the cache), attacker can determine if the victim accessed that cache line or not during its normal execution. Since, attacker does not make any extra cache accesses/reads, it is harder to track this type of attack. That's why Flush+Flush is considered to be a stealthy attack.



(a) Victim and attacker sharing same address space (b) Flushing cache line (c) Reloading line to observe timing

FIGURE 6. Working principle of Flush+Reload.



(a) Victim and attacker (b) Flushing and timing when victim is executing (c) Flushing and timing when victim is not executing

FIGURE 7. Working principle of Flush+Flush.

Moreover, this type of attack is very fast. However, it is thought of having more noise and normally results into higher error rates compared to F+R and P+P attacks [5], [14], [30].

C. MITIGATION OF CACHE SIDE CHANNEL ATTACKS

Various research works have focused on the mitigation of cache side channel attacks in the recent past. Some of these techniques focus on resource isolation by partitioning caches either through software or in the hardware [31], [47], [47], [48], [48]–[52]. Partitioned portions of the cache are essentially reserved for the protected program and this non-interference avoids CSCAs to work. Due to cache partitioning / reservation, these techniques impose significant performance bottlenecks. Further, they also increase hardware overheads due to the need of specialized hardware features like in [50]. Another technique that has been proposed as a defense against CSCAs is the randomization of the mapping between memory and cache sets [53]. This way same addresses of different applications can be mapped to different sets in the cache thus thwarting possibility of CSCAs. Another approach to mitigate CSCAs is the addition of noise either in HPCs or clock sources (used for time measurements). This has been done both in the software [54]–[56] and hardware [57].

Although a lot of research efforts have been done to propose novel mitigation techniques against malicious side-channel attacks, such techniques still need improvements. Mitigation techniques generally focus on a specific vulnerability and don't provide an all-weather protection as it can be expensive and complex. At the same time, there has been a continuous progress in the domain of attacks which keep on getting complicated and stealthier. Therefore, the gap between the demands of a CSCA mitigation technique and what they offer is increasing as well. We argue that in this scenario, CSCA detection techniques can work in synergy with CSCA mitigation and prevention techniques to simplify their design and performance cost. CSCA mitigation and prevention techniques would be activated only if a detection technique raises an alarming flag. CSCA detection techniques have to be accurate and fast in order to be useful when coupled with CSCA mitigation techniques. Researchers have proposed various techniques to detect cache based side channel attacks. It is important to understand the existing CSCA detection mechanisms and identify any improvements that can be done. Despite the importance of CSCA detection techniques, there does not exist any work that tried to survey current CSCA detection techniques. Our paper discusses the details of state-of-the-art CSCA detection techniques and compares them using a common set of parameters.

D. RELATED SURVEYS

There exist a few survey papers which review side channel attacks in general or cache side channel attacks in specific. However, we have not been able to find any work that focuses on the review of research works that performed run-time detection of cache based side channel attacks (CSCAs). Szefer [58] broadly reviewed various microarchitectural side channel attacks along with their defense mechanisms that have been proposed so far. Ge *et al.* [13] also surveyed microarchitectural side channel attacks specially in cloud systems and discussed future trends and possible solutions for such attacks. Ge *et al.* [30] enlist a variety of cache side channel attacks and mitigation strategies specifically on modern hardware. Similarly, [59] discusses different software and hardware attacks and also studies the performance overheads of different measures against attacks specifically for AES crypto algorithms. Lyu and Mishra [60] studied CSCAs on famous encryption algorithms alongwith their countermeasures. Kong *et al.* [52] studied hardware-software techniques for a defense for CSCAs. Jin [61] discussed the basic concepts of hardware security in general. Zhou and Feng [62] surveyed various approaches to perform side channel attacks and their mitigation techniques. Younis *et al.* [63] provided an extensive analysis of CSCAs in cloud systems and compared few CSCA mitigation and detection mechanisms. The comparison is performed using three categories of CSCAs (Prime+Probe, Flush+Reload and Flush+Flush). The CSCA detection techniques that were evaluated and compared in detail include HomeAlone detection solution [64] and two-stage detection technique [10]. We will discuss these techniques in detail later on.

III. FACTORS USED FOR COMPARISON OF CSCA DETECTION TECHNIQUES

We identify a number of important factors that can be used to compare and characterize the proposed CSCA detection techniques. It should be noted that this list is not exhaustive, but a list of the most important factors that we recognized based on the studied CSCA detection techniques.

A. DETECTION ACCURACY

Detection accuracy should be considered to be the primary metric to judge any intrusion detection mechanism. Since, detection of side channel attacks is a binary classification problem, detection inaccuracy can be further divided into false positives (cases when a no-attack condition is detected as an attack) and false negatives (cases when an attack condition is detected as no-attack) to analyze detection results in details. Two of the commonly used metrics that have been used to represent detection accuracy in the reviewed literature are Percentage Accuracy and F-score. The reason for using F-score often over percentage accuracy is following: F-score is generally not influenced by data sets in which one class might have much more number of samples (also known as *skewed class*) than the other classes.

B. DETECTION SPEED

The speed with which an attack is detected is another important indicator for evaluating any detection proposal. Detection speed is usually a trade-off between overhead of a detection system and timely intrusion detection. Detection speed is a function of the crypto-system (which the attack is targeting) and the attack itself and should be considered accordingly. For example, Flush+Reload on RSA is a single-encryption round attack and for detection to be useful the attack should be detected before half of the total bits are encrypted. On the other hand, Flush+Flush on AES requires hundreds of encryption rounds to be successful, so its detection can be useul even if it is done after number of encryption rounds. Few of the mostly used metrics to indicate detection speed in literature include: the absolute time, the number of encryption rounds being performed by the crypto-system and the number of bits being encrypted by the crypto-system by which a detection mechanism is capable of detecting an attack.

C. DETECTION OVERHEAD

Nothing is free. The detection mechanism will always incur some performance overhead depending on its complexity and the level of implementation. Detection overhead can be defined as the slowdown of the process to be protected due to the implemented detection mechanism. The detection overhead will be determined by the detection granularity which specifies how often the detection mechanism would be activated to make a decision based on the information provided. Secondly, the perceivable detection overhead is related to the implementation of the detection mechanism as well.

It is important to note that in this work we mainly consider the runtime detection overhead. There exist other types of overheads as well like time to train and implement a particular detection strategy. However, such overheads are usually one-time (or rare) and can be considered insignificant in comparison to runtime detection overhead.

D. USED ATTACKS

There are various techniques to perform cache side channel attacks (CSCAs) as discussed in the previous section. The difficulty of detection of a CSCA varies depending on the used technique and the crypto-system under consideration. For example, Flush+Flush attack is considered to be more stealthier compared to Flush+ Reload attack [14]. Therefore, in order to perform a comparison of CSCA detection techniques it is essential to identify particular attack techniques along with the crypto-systems that were used to evaluate the working of a detection mechanism for CSCA.

E. IMPLEMENTATION LEVEL

CSCA detection mechanism can be implemented at different levels in a computer system. The possible implementation levels include: victim application (crypto-system) itself, as a separate application/process, within the operating system,

inside each Virtual Machine (VM) or directly inside the hardware. It is important to compare CSCA detection techniques based on how they are physically implemented as each level of implementation will have its own strengths and weaknesses. For example, implementing a CSCA detection mechanism as a separate application can be slow, but such a solution can work with legacy systems/hardware. On the other hand, a detection mechanism implemented inside the hardware will be fast but will not be portable to legacy systems/hardware.

F. USE OF HARDWARE PERFORMANCE COUNTERS (HPCs)

Hardware performance counters (HPCs) are special purpose hardware registers available in most of the modern processor families. HPCs are used to monitor performance of applications by counting a number of microarchitectural and architectural events (like cache misses, executed instructions, branch mispredictions etc.) during the application's execution on the hardware. Intel x86 based processors [65] provide access to hundreds of hardware events that can be tracked using HPCs. However, due to limitation of the number of physical registers, only a few of the events can be monitored simultaneously (usually 4 to 8). Different libraries and APIs like PerfMon [66], OProfile [67], Perf tool [68], Intel Vtune Analyzer [69], and PAPI [70] can be used to configure and read HPCs. Most of the CSCA detection mechanisms have tried to use HPCs in their detection algorithms. Therefore, it is significant to observe how do the used HPCs differ across CSCA detection approaches.

G. USE OF MACHINE LEARNING (ML) MODELS

Machine learning techniques have influenced many fields and are gaining popularity in the field of information security as well [71]. Differentiating an attack from a non-attack case is basically a classification problem. Many side-channel detection techniques have used ML models (classifiers) to distinct attacks and non-attacks. Majority of the supervised machine learning classifiers fall into two classes of linear and non-linear models. Commonly used ML classifiers in CSCA detection techniques inlcude: Linear Discriminant Analysis (LDA), Logistic Regression (LR), Support Vector Machine (SVM), Random Forest, Decision Tree, K-Nearest Neighbor (KNN) and Neural Networks (NN). Details of these ML models can be obtained from these soruces: [72]–[74]. Each machine learning algorithm has different structure and will have different computational and memory cost for training and performing an inference decision based on the trained model. Therefore, analysis of types of ML models used in a side-channel detection technique becomes an important parameter to compare different detection techniques.

H. DESIGN CATEGORY

Most of the side-channel detection techniques can be divided into two basic categories based on their design: Signature based detection and Anomaly based detection. Signature based detection approaches rely on signature of "known side-channel attacks" which usually consist of selected hardware events that will be affected by those attacks. At run-time, program execution is compared with the already generated signatures and in case of a match an attack is detected. Such detection approaches usually show very good accuracy in detection of known attacks [75]. However, they might suffer from low accuracy for detection of unknown or modified attacks [75]. Anomaly-based detection approaches generate model of the behavior of normal/benign applications. Any significant "deviation" from such model will be considered an attack. Anomaly-based detection techniques are capable of identifying unknown or modified attacks [75]. However, they can have high false positive rates [75] as it is hard to build models including every possible benign application and many benign applications can resemble cache based side channel attacks due to their high memory usage.

Some research works [75], [76] have also combined both anomaly and signature based detection designs to achieve better results.

I. OTHER FACTORS

A couple of other factors that can prove to be valuable while comparing different detection techniques include: *Load/Noise:* This refers to the use of load/noise in the background while evaluating the proposed attack detection mechanism. It is important for the robustness of a detection technique to perform well when the system is under load (as would be the case most of the times, when attacks take place). *Attacker Identification:* This checks if the detection mechanism can also identify the actual process performing the attack. If the attacker is also identified it makes it much easier for mitigation techniques to deal with system under attacks.

IV. SURVEY OF CSCA DETECTION TECHNIQUES

This section discusses in detail the surveyed literature about detection of cache based side channel attacks. Table 2 presented in next section summarizes and shows a comparison of all surveyed CSCA detection techniques on a common set of parameters. The reader should use special care while comparing different detection techniques based on the metrics (of Table 2) as the point of reference of some of these metrics (like detection accuracy, detection speed) could be different. For example, the reported detection accuracy of two techniques can be based on different attacks. As discussed in the previous section, CSCA detection techniques either employ anomaly based detection, signature based detection or a combination of both. Majority of the CSCA detection techniques are signature based techniques: [77]–[85] or a combination of both signature and anomaly based techniques [75], [76], [86]. Figure 8 shows a classification of CSCA detection techniques based on their fundamental design type.

TABLE 2. Summary of the reviewed CSCA detection techniques.

Reference	Category	Example Attacks	Detection Accuracy	Detection Speed	Detection overhead	ML-Based	Attacker Identification	Use of HPCs	Impl. Level	Load/ Noise	Crypto System
Demme et al [77]	Signature Based Detection	P+P	100%	N/A	N/A	Yes (KNN, DT, RF and ANN)	Yes	Yes	Application & Hardware	N/A	
Allaf et al [78]	Signature Based Detection	F+R & P+P	97% (F+R), 98% (P+P)	2% of time to complete attack	N/A	Yes (NN, DT, KNN)	Yes	Yes	Yes	Yes	AES
Allaf et al [79]	Signature Based Detection	F+R	99% (Native), 96% (Cloud)	N/A	N/A	Yes (KNN)	Yes	Yes	Application	Yes	AES
Mushtaq et al [85]	Signature Based Detection	F+R & F+F	> 99% (both attacks)	1% of RSA, 12.5% of AES completion	< 2%	Yes (LDA, LR, SVM)	No	Yes	Application	Yes	RSA & AES
Mushtaq et al [99]	Signature-Based Detection	F+R & F+F	> 99% (both attacks)	0.9% of F+R-RSA, < 40% of F+R-AES, < 25% & < 0.8% of 2-F+F-AES	< 2 for RSA and < 10% for AES completion	Yes (LDA, LR, SVM, QDA)	No	Yes	Application	Yes	RSA & AES
Mushtaq et al [97]	Signature-Based Detection	P+P	> 99%	1-2% of AES completion	< 3-4%	Yes (LDA, LR, SVM, QDA)	No	Yes	Application	Yes	RSA & AES
Mathias Payer [153]	Signature Based Detection	F+R & P+P	100%	N/A	< 2%	No	Yes	Yes	Kernel	No	
Peng et al [81]	Signature Based Detection	F+R	100%	N/A	N/A	No	Yes	Yes	Application	No	
Briongos et al [82]	Signature Based Detection	F+R	>96%	N/A	N/A	No	N/A	Yes	Application	No	AES
Raj and Dharanipragada [84]	Signature Based Detection	P+P, F+R	N/A	N/A	<8%	No	No	Yes	VM	No	

Note: N/A: Not Available/Applicable, HPC: Hardware Performance Counter, P+P: Prime+Probe, F+R: Flush+Reload, F+F: Flush+Flush, E+T: Evict+Time, VM: Virtual Machine, CVM: Cross Virtual Machine, Impl:Implementation ML: Machine Learning. Also, note that the mentioned detection accuracy,speed and overhead are the best-case measures for each technique

1) SIGNATURE-BASED DETECTION TECHNIQUES

One of the earliest CSCA detection works was done by Demme *et al.* [77] who tried to detect malware and CSCAs based on signatures. Demme *et al.* [77] claimed to use hardware performance counters (HPCs) for the first time to solve the problem of malware and side-channels detection.

TABLE 2. *(Continued.)*

Reference	Category	Example Attacks	Detection Accuracy	Detection Speed	Detection overhead	ML-Based	Attacker Identification	Use of HPCs	Impl. Level	Load/ Noise	Crypto System
Chen et al [107]	Signature Based Detection	Reference Clock Attack, Application Thread Attack and CPU Speed Manipulation	Precision: 0.83 (Ref. Clock), 0.95 (App. Thread), 0.96 (CPU Speed Manipulation)	N/A	<5%	No	No	No	extension of LLVM [157]	N/A	
Chouhan et al [83]	Signature Based Detection	F+R	100%	≈35% of time to execute attack	N/A	No	N/A	No	VM	No	
Paundu et al [110]	Signature Based Detection	P+P, F+R, F+F	0.99 (AUC)	N/A	0.7%	Yes (SVM)	Yes	Yes	VM	Yes	
Yu et al [64]	Signature Based Detection	P+P, E+T	N/A	N/A	N/A	No	No	Yes	VM	Yes	
Bazm et al [112]	Anomaly Based Detection	P+P	100%	N/A	2%	Yes (Gaussian Anomaly Detection)	N/A	Yes	VM	Yes	
Briongos et al [113]	Anomaly Detection	F+F, P+P, F+R	100%	within 37% of ElGamal and 50% of RSA execution	N/A	Yes	No	Yes	Application	Yes	AES, ESA & ElGamal
Kulah et al [114]	Anomaly Based Detection	P+P, F+R, F+F	0.93 for P+P (phy. & CVM) 0.99 & 0.97 for F+R (phy. & CVM), 0.82 & 0.96 for F+F (phy. & CVM)	N/A	0.49-3.58%	Yes (Anomaly Detection)	Yes	Yes	VM	Yes	AES & ECDSA

Note: N/A: Not Available/Applicable, HPC: Hardware Performace Counter, P+P: Prime+Probe, F+R: Flush+Reload, F+F: Flush+Flush, E+T: Evict+Time, VM: Virtual Machine, CVM: Cross Virtual Machine, Impl:Implementation ML: Machine Learning. Also, note that the mentioned detection accuracy,speed and overhead are the best-case measures for each technique

TABLE 2. ***(Continued.)***

Reference	Category	Example Attacks	Detection Accuracy	Detection Speed	Detection overhead	ML-Based	Attacker Identification	Use of HPCs	Impl. Level	Load/ Noise	Crypto System
Chiappetta et al [86]	Anomaly + Signature Based Detection	F+R	F-Score: 0.93 (AES), 1.0 (ECDSA)	1/5th of attack completion	N/A	Yes (Anomaly Detection, Neural-Network)	Yes	Yes	Application	Yes	AES & ECDS
Zhang et al [75]	Anomaly + Signature Based Detection	P+P, F+R	100%	order of ms	$< 5\%$	No	N/A	Yes	VM	N/A	Both symmetric and asymmetric
Alam et al [76]	Anomaly + Signature Based Detection	[145], [146]	>99%	N/A	N/A	Yes (RF, SVM, Adaboost, Perceptron, NB)	Yes	Yes	Application	Yes	AES & Clefia
Zhang et al [137]	Signature Based Co-location Detection	P+P	85%	N/A	$< 4.6\%$	No	Yes	Yes	VM	Yes	
Inci et al [138]	Signature Based Co-location Detection	P+P, F+R	93% & 90%	N/A	6.1x	No	No	No	commercial clouds (Amazon EC2,Google Cloud engine, Microsoft Azure	N/A	RSA & AES
Wang et al [148]	Signature Based	F+R, F+F	N/A	N/A	4.7%	No	Yes	Yes	Application	Yes	

Note: N/A: Not Available/Applicable, HPC: Hardware Performace Counter, P+P: Prime+Probe, F+R: Flush+Reload, F+F: Flush+Flush, E+T: Evict+Time, VM: Virtual Machine, CVM: Cross Virtual Machine, Impl:Implementation ML: Machine Learning. Also, note that the mentioned detection accuracy,speed and overhead are the best-case measures for each technique

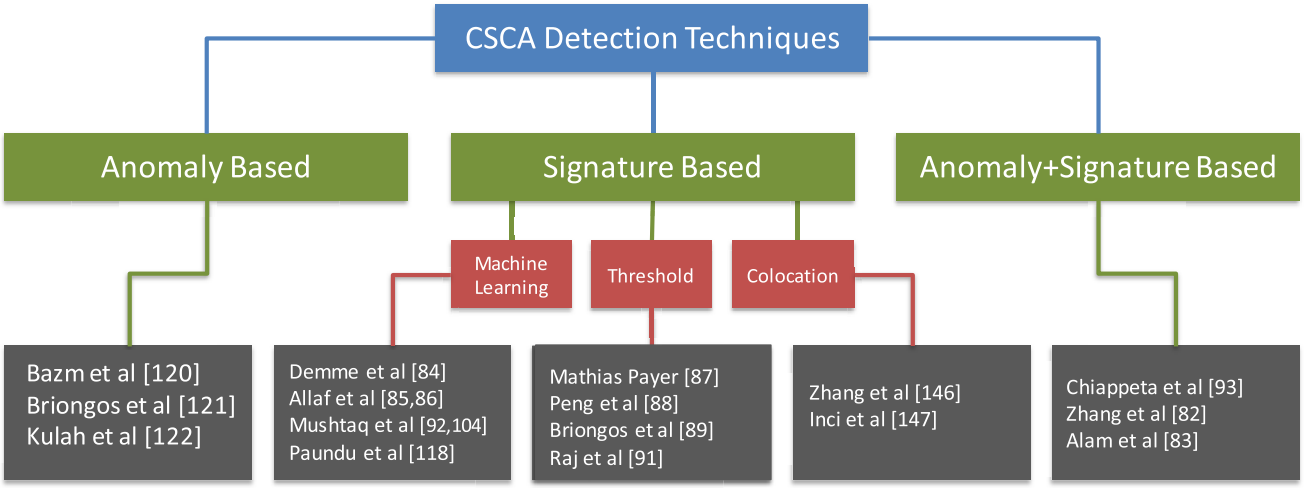


FIGURE 8. Classification of CSCA detection techniques.

The proposed machine learning (ML) based detection technique relies on performance counters which are sampled at regular intervals and used as input feature vectors to ML models/classifiers. The particular ML classifiers used in this work include: K-Nearest Neighbor (KNN) [87], Decision Trees [88], Random Forest and Artificial Neural Networks (ANN) [89]. Demme *et al.* [77] used a case study of Prime+Probe (P+P) type of side channel attacks to evaluate their proposed CSCA detection mechanism. Importantly, their proposed detection mechanism is able to detect the attack process as well. Their experiments relied on OpenSSL as a victim process and a set of benign processes from different benchmark suites SPEC-CPU2006 [90], PARSEC [91] and other applications like games, web browsers and graphics editors. They used variants of Prime+Probe attack to generate training and evaluation data for ML algorithms. Results show that all of the used ML models (KNN, Decision Trees, Random Forest and ANN) are able to detect all of the attack threads without any false positives. The authors also presented a possible hardware implementation of their detector which would consist of four basic blocks: Data Collector, Data Analyzer, Action System and Secure Updater. The possible design choices to implement the detector module in hardware include: separate cores, accelerators, μ-controller, FPGA or a coprocessor [77]. This is the only work out of the surveyed research works that proposed a possible hardware implementation of their detection module.

Another approach named *SCADET* [92] is a signature-based detection tool, which detects Prime+Probe attack. Instead of using HPCs (hardware performance counters), this approach uses high-level semantics and invariant patterns of attack (targeting I-cache, D-cache and LLC). Results show that *SCADET* provides high accuracy but they lack discussion on detection speed and performance overhead of the mechanism. Authors report that, in some cases, system provides false alarms under load conditions. Moreover, the trace analysis time in this approach is very long (notable irregularities when trace exceeds a certain size), which renders the solution not suitable for run-time detection.

Allaf *et al.* [78] also presented a signature based CSCA detection mechanism that uses Machine Learning (ML) to generate signatures which are representative of attacks. Allaf *et al.* [78] used three ML algorithms namely Neural Networks [89], Decision Trees [88] and K-Nearest Neighbor (KNN) [87] to detect cache based side channel attack specifically on AES crypto-system. The particular side channel attacks used in their work are Flush+Reload (F+R) and Prime+Probe (P+P). A data set containing values of seven different hardware performance counters which include core cycles, reference cycles, core instructions and other four features having the best effect on classification of attack and no-attack scenarios for the used attacks is collected during execution of processes (attacks and benign processes). This data set is used for both training and validation of machine learning algorithms. The data set covers two scenarios: with and without any noise in the background when attacks and victim programs are running. Integer and Floating-point categories of SPEC-CPU2006 benchmarks (SPEC-int and SPEC-fp) [90] are executed in the background to simulate noise/load conditions.

Allaf *et al.* [78] also processed training data before using it to train ML classifiers. The dimensions of training data are first reduced using a technique of Principal Component Analysis (PCA) [93]. The data is then passed through a well-known optimization algorithm called L-DFGS [94], which is known for its affinity towards smaller data sets. The particular decision tree used in their work is C4.5 [95] which is a famous tree-based statistical classifier. Evaluation of the proposed technique on an Intel Xeon (X5650) processor shows that the best classification success rate is shown by Decision Tree which is 0.97% for F+R and 0.98% for P+P attacks in case of no SPEC benchmarks in the background.

The accuracy is reduced in case of background SPEC benchmarks (specially in case of SPECfp benchmarks, which according to authors' claim make heavier use of CPU components specially caches compared to the integer benchmarks). However, decision trees still have better accuracy compared to other methods under noisy conditions. Results further show that the detection framework (which learns at run-time) is able to learn the behaviour of malicious process in less than 1 second in worst case, which authors claim is very fast in comparison to 50 seconds required for retrieval of entire key bits by an F+R attack implementation done by [96] on the used machine. Results also show that decision trees are less efficient (have low detection speed) compared to the other methods but show better accuracy.

Later, Allaf *et al.* [79] used Machine Learning to generate signatures of malicious loops of attack processes to detect them at run-time. Allaf *et al.* [79] specifically used K-Nearest Neighbor (KNN) classifier to detect malicious loop activity within Flush+Reload (F+R) attack do detect attacks without the need of observing any synchronization between attacker and victim process as some other techniques [75], [76] do. The used machine learning model is trained using three features of L1, L2 and last-level cache (LLC) misses. Selected benchmarks (*bzip2, gcc, bwaves, dealII*) from SPEC-CPU2006 [90] benchmark suite are executed in the background to create realistic system conditions. Programs are profiled by reading performance counters at time intervals of 0.02μs, which is the time that a single run of malicious loop of F+R attack takes. N number of profiled samples are grouped together and are represented by the average of those samples. These representative samples are fed to K-NN classifier to make classification decision. The experimental evaluation of the presented model on an Intel Xeon system shows that it can achieve an accuracy of 99% on native system and 96% on a cloud system, without any extra overhead on cloud system. However, authors declare that this mechanism would not work for other attacks like Prime+Probe considering the differences in working of malicious loop in the attack. Moreover, authors claim that the trained classifier does not need to be re-trained to detect hostile processes in a new environment.

Some of the research works belonging to the category of signature-based detection that have been done recently include: [85], [97]–[99]. One of these works done by Mushtaq *et al.* [85] targets stealthier CSCAs like Flush+Flush (F+F). Mushtaq *et al.* [85] proposed *NIGHTs-WATCH* to detect cache-based side-channel attacks at run-time using ML models (LDA, LR, SVM). Different hardware performance counters are used to profile victim cryptosystems RSA and AES under attack and no-attack scenarios to train ML models. *NIGHTs-WATCH* is embedded into the cryptosystems to profile them at run-time using HPCs and apply trained ML models to detect presence of any side-channel attacks. The particular case studies used to evaluate *NIGHTs-WATCH* include Flush+Reload and Flush+Flush attacks. *NIGHTs-WATCH* being a run-time detection mechanism is evaluated using a variety of metrics like detection accuracy, speed and overhead. Evaluation of the proposed detection technique shows that it can achieve a high detection accuracy with little performance overhead for both attacks even under noisy conditions. Specifically, it is shown to have a detection accuracy of 99.51%, 99.50% and 99.44% for Flush+Reload attack under no, medium and high noise at very high detection speed (within 1% completion of a single RSA encryptions). For Flush+Flush attack, the detection accuracy is shown to be 99.97%, 98.74% and 95.20% for no, medium and high noise at high detection speed (within 12.5% completion of 400 AES encryptions which is the minimum number of encryptions required for a successful attack). Performance overhead of the detection module is less than 2% in both cases. Later on, this work was extended by Mushtaq *et al.* [97], [98] to include Prime+Probe and other variant attacks under both RSA and AES cryptosystems. These approaches also use ML models (LDA, LR, SVM, QDA) and HPCs to detect different variants of these attacks at run-time. Reference [98] shows a high detection accuracy of up to 99.51% for Flush+Reload attack on RSA, incurring a performance overhead of 1.63% and 99.99% accuracy on AES while incurring a maximum performance overhead of 8.28%. The experimental results show consistency for Flush+Flush attack on different implementations of AES as well. Reference [97] show detection accuracy of $>$ 99% for Prime+Probe attack and their variants running on AES crypto-system with performance overhead of $3-4\%$ at the highest detection speed.

Some of the signature based detection techniques don't rely on Machine Learning to learn attack signatures. Rather they use thresholds of particular hardware events to determine if an attack is in place. Examples of such works include: [80]–[82], [84]. One of these works, done by Mathias Payer [80], utilizes the values of cache miss rates and page faults of processes to detect an attack. Mathias Payer [80] proposed an attack detection framework *HexPADS* which can detect cache based side channel attacks along-with Rowhammer [100] and CAIN [101] attacks. *HexPADS* reads status of different performance counters like total executed instructions, total LLC accesses and total LLC misses. It also uses kernel information of processes like total page faults. Same type of detection technique is used for both row-hammer and cache side channel attacks and does not distinguish between the two. The proposed detection mechanism basically continuously monitors the cache accesses and misses of all processes. If cache miss rate of a process is found to be higher than 70% i.e. greater than 70% of cache accesses results into misses, and the same process has a low number of page-faults, the process is detected to be an attack. The evaluation of the proposed detection technique is done using following attacks: cache template attacks [15] based on Flush+Reload (F+R) and an enhanced version of C5 [102] based on Prime+Probe (P+P) attack. Performance overhead of the detection framework is measured by executing SPEC-CPU2006 [90] and PARSEC [91] benchmark suites when

detection framework is active, indicating that the mean of overhead (loss in performance of executed benchmarks) is less than 2%. Experiments show that *HexPADS* can detect both attacks successfully. However, it is not evaluated using any realistic load/noise conditions.

Another threshold based technique which is similar to the one proposed by Mathias Payer [80] has been presented by Peng *et al.* [81]. Peng *et al.* [81] used cache miss rates and data-TLB miss rates to recognize cache side-channel attacks. They showed that cache side channel attacks like Flush+Reload (F+R) have high cache miss rates but low dTLB (Data Translation Lookaside Buffer) miss rates. The detection mechanism scans all running processes on a system and observe the values of performance events specifically cache and dTLB miss rates for these processes. A detection flag is raised if the cache miss rate is found to be above and dTLB miss rate below a particular threshold. Variants [5], [15], [102] of Flush+Reload type of CSCA are used for evaluation of the proposed technique. Experimental results show that this technique is able to discriminate cache based side channel attacks from benign processes and other timing attacks accurately. Experimental analysis does not present other run-time detection evaluation metrics like speed and overhead.

The work of Briongos *et al.* [82] also depends on the comparison (of encryption times) with set thresholds to determine the occurrence of CSCA. Briongos *et al.* [82] built a timing model to discriminate if a process is being attacked or not. Cache based side channel attacks on AES encryption system are considered in this work. As shown in [82], the distribution of AES encryption times under attack and no-attack cases shows an observable distinction when no other processes are executing on the CPU. Authors conclude that in such a case encryption times above a threshold would be highly indicative of an attack. To create a realistic scenario, authors experimented with running of other workloads in the background along-with an attack. The first case involves running *Lookbusy* program [103] in the background, which is a CPU-centric workload designed to stress computational capability of a processor. The distribution of encryption times in this case shows that the peak heights indicating non-attack cases rise. In the second case, a memory benchmark *RandMem2* [104] is used to stress memory system by performing random accesses to memory. The results in this case showed that this process only caused a single cache miss for one encryption at maximum and affects lower than 1% of encryption rounds. This infers that the CPU consumption will have more effect on time distribution (which is to be used in the detection process). Based on these observations, the proposed cache side channel attack detection algorithm uses the time distribution of encryption algorithm. The method uses last 200 samples of encryption times at any time instance. From these samples, a histogram is created (using time intervals of 20 cycles). Peaks of this histogram are found using a windowing operation. The height of these peaks are used to decide if an attack is active or not. Experimental results show that the proposed detection algorithm achieves a detection accuracy greater than 96% (false positive rate of 5%). It is shown that the false positive rate can be further reduced to 0% if initializing stage of victim process is ignored.

Raj and Dharanipragada [84] presented *PokerFace* to identify and mitigate cache attacks which compares the memory bus bandwidth with a threshold level to detect a CSCA. The proposed framework consists of two components: Poker and Face (both are implemented as single threads in guest VM). Poker is responsible for detection of attacks which triggers Face upon an attack detection. Face then performs cache obfuscation to make the attack unsuccessful. The attacks are detected at the level of VM. Poker works by observing memory bus bandwidth to obtain information regarding cache accesses. The working of Poker is based on the fact that during a cache attack the victim VM suffers from significant degradation in memory bus bandwidth. The evaluation of the proposed framework is done using Prime+Probe and Flush+Reload types of cache side channel attacks. Performance overhead of *PokerFace* using STREAM [105], Sysbench [106] and PARSEC [91] benchmark suites is found to be less than 8%.

Intel introduced an extension to their instruction set architecture (ISA) named Intel Software Gaurd Extension (SGX) to protect the execution of unprivileged programs inside secure enclaves. Still the privileged programs with malicious intent can perform side channel attacks on programs inside a secure enclave. The work of Chen *et al.* [107] employs a threshold based design to detect a special case of cache side channel attacks. Chen *et al.* [107] proposed *Deja-Vu* to detect side channel attacks on programs guarded by SGX. Privileged attacker regularly preempts the shielded execution of victim process which is executing inside an enclave. This leads to unanticipated enclave exits which are known as Asynchronous Enclave Exits (AEXs). These preemptions can be observed by the operating system (OS) and a higher frequency of such preemptions indicate the presence of an attack. *Deja-Vu* detects the existence of AEXs to identify the presence of an attack. *Deja-Vu* needs a reference clock, that cannot be compromised, to measure the execution time of SGX application to be protected. The execution time of the application at run-time when detection mechanism is active is compared with the normal run-time (run-time of the process when no-attack is in place). A time difference above a threshold indicates the possibility of enclave exits and a possible attack. To make sure that the used reference clock is trust-worthy, it is protected by Intel's hardware transactional memory (TSX) support. The run-time overhead of *Deja-Vu* is found to be less than 5% using nbench [108] benchmark suite. However, the required instrumentation can increase the size of enclave binaries by approximately 64%.

An example of the signature based CSCA detection approaches that uses special data structures (like bloom filters) is the work of Chouhan and Hasbullah [83]. Chouhan and Hasbullah [83] used bloom filters [109] to propose a detection for cache based side channel attacks in

cloud systems. The use of bloom filters is motivated by the need to reduce the performance overhead of the detection mechanism. Chouhan and Hasbullah [83] fed cache miss time mean values read from performance counters to bloom filters which detect if the values belong to an attack condition or not. Bloom filters, due to their fundamental nature, do not lead to any false negatives. For readers not familiar with bloom filters, bloom filters are used to decide if a certain element is a member of a particular set. Once, certain index values are generated after an hash function is computed on elements of set under consideration, bits in bloom filter corresponding to those index values are set to true. For any new element (for which decision about its membership to the set under consideration is to be made), it is passed to the hash functions and it's seen if bit indexes corresponding to hash functions' outputs are set to 1 (membership would be true if outputs are set to 1 and false otherwise). Bloom filters prevent false negatives, but can lead to cases of false positives. Bloom filters are supposed to be very efficient to find memberships of elements in a set as they don't rely on actual comparisons, rather use hash functions.

The proposed detection technique of Chouhan and Hasbullah [83] first records the cache miss patterns of processes with the help of performance profiling tools like *perf*. Cache miss times (CMT) for these patterns are also calculated with the help of a timer. Mean of the differences of each successive CMT is calculated and formed signatures are stored in a bloom filter. At run-time, the detection mechanism calculates such signatures again and pass them to bloom filter to check membership of the signatures under consideration. If set membership is found to be true, it indicates a high probability of an attack. The methodology is evaluated with the help of a cache simulator. Experimental results indicate that the proposed solution takes around 6 seconds to execute on the used machine in comparison to 17-25 seconds required to execute the Flush+Reload attack. The authors claim that the proposed mechanism should also work for the detection of unknown cache based side channel attacks.

Signatures based on KVM (kernel virtual machine) events have also been used in the detection of CSCA. Paundu *et al.* [110] proposed a CSCA detection technique in a virtualized environment using the information of KVM (Kernel Virtual Machine) events. These events (collected using ftrace utility [111]) provide information about the host kernel operations when a guest system is running on it (i.e. they monitor the guest activity). A machine learning model SVM (with RBF kernel) is trained using the KVM events data for specific time sequences. A set of normal applications (including idle VM, web and mail server applications) forms the no-attack data set needed to train the SVM. Experimental evaluation shows a performance overhead of 0.7% for a host system based on Intel Xeon processor (set up with 8 VMs). All three classes of CSCA techniques (Prime+Probe, Flush+Reload and Flush+Flush) are used to evaluate the proposed CSCA detection technique. ROC (Receiver Operating Characteristic) curve of the trained classifier shows an AUC (Area Under the Curve) value of 0.99 while classifying attack and no-attack scenarios.

Yu *et al.* [64] also presented a signature based two-stage CSCA detection technique named as *CSDA* (Cache-based Side Channel Attack Detection Approach) with a focus on cloud systems. The two stages of *CSDA* include detection at the level of host and guest respectively. *CSDA* makes use of shape and regulatory tests which are significant methods used to analyze detection in covert channels. Shape tests utilize first order statistics like mean, variance and entropy to describe different features. Regulatory tests utilize second order or higher statistics like correlations and mutual information found in data. In *CSDA*, at the level of host detection, shape tests are executed to reveal the features of attack using CMS (cache miss sequence). Whereas, guest detection is the second phase which is dependent on the results of host detection. During guest level detection, regulatory tests are conducted to obtain the features of attack which are extracted from virtual CPU and memory utilization. Two-stage detection technique of Yu *et al.* [64] extracts the features of attack from host and guest and then uses pattern recognition techniques to distinguish attacker VMs from non-attacker VMs. Experiments reveal that *CSDA* is able to detect malicious VMs efficiently in cloud setup. Whereas, no empirical results on performance overhead, detection accuracy or detection speed has been found in this paper.

2) ANOMALY-BASED DETECTION TECHNIQUES

There are a few recently proposed research works [112]–[114] that rely solely on anomaly detection for recognition of CSCAs. Bazm *et al.* [112] relied on *Intel Cache Monitoring Technology (CMT)* [115] and hardware performance counters and used Gaussian Anomaly detection [116] for detection of cache based side-channel attacks at the level of VMs in Iaas Cloud platforms. The proposed mechanism shows very good accuracy in isolated conditions but suffer from high false positives in noisy conditions. Intel Cache Monitoring Technology (CMT) provides ''fine-grained'' information of behavior of caches in virtualized environment. CMT also monitors the use of shared resources such as last level caches (LLC) in modern processors and provides statistics like occupancy of LLC by VMs on a particular physical machine. The information provided by CMT can be used to improve the detection of side-channel attacks in VMs. The proposed approach to detect cache side channel attacks uses some hardware performance counters (LLC misses and references, iTLB cache misses and accesses) along-with the information provided by CMT. The model used for detection of anomalies i.e. Gaussian Anomaly Detection is trained on the data of counters by estimating Gaussian distribution of all features (after calculating their mean and variance). Each virtual machine on the physical host acts as a single data-point in this work and values of performance counters and LLC-occupancy act as features of that data point.

The proposed framework of Bazm *et al.* [112] consists of multiple threads: first thread probes the entire system to gather statistics (performance counters and LLC occupancy) of all VMs, second thread provides a list of active VMs. Third thread runs Gaussian anomaly detection using the gathered statistics to make any detection decisions. The proposed framework is evaluated using an experimental system based on Intel Xeon running 6 VMs. The particular cache side-channel attack used in their work is an implementation of Prime+Probe provided by [14], [117]. Moreover, four different scenarios are built using an attacker VM, a victim VM and a CIW VM (VM running compute-intensive workloads): (1) No Attack, No-CIW, (2) Attack, No-CIW (3) No-attack, CIW (4) Attack, CIW. Experimental results indicate that detection module is able to perform detection with absolute accuracy for first two scenarios. However, it can result into false positives for cases (3) and (4) with CIW VMs. Experiments further indicated that iTLB cache miss rate vary significantly for attacker and CIW VM (attacker VM shows low iTLB-cache miss rate while CIW VM shows high iTLB-cache miss rate) and can be used to improve the accuracy of cases (3) and (4) by reducing false positives. This finding is similar to what Peng *et al.* [81] and Payer [80] found as well. Further, it is shown that the proposed detection module incurs around 2% performance overhead to the hypervisor. However, this overhead might increase with an increase in the number of VMs.

Briongos *et al.* [113] proposed *CacheShield* to detect cache side channel attacks on legacy software (victim applications) by monitoring hardware performance events during their execution. The proposed method is implemented at user level and does not require any help from the OS/hypervisor and would be applicable in cloud environments. As indicated by the authors, this effort is motivated by two main problems of the other detection mechanisms: high detection performance overheads for VMs and requirement of monitoring of both attacker and victim at the same time. The proposed attack detection technique, *CacheShield*, uses an unsupervised anomaly detection algorithm *Cumulative Sum Method (CUSUM)* proposed by Page [118]. *CUSUM* belongs to the category of change point detection (CPD) [119] algorithms. CPD algorithms determine when their is a major change in the characteristic parameters of the system under consideration. Using the infoGain function of WEKA tool [120] and relief algorithm [121] on a number of hardware performance events collected for RSA crypto algorithm under attack, Briongos *et al.* [113] found that the most relevant/meaningful event is L3 cache misses for detection of attacks. *CacheShield* monitors performance counters in parallel to any victim application making it possible to detect attacks that will be successful during working of a single call of the ''sensitive function''. Using two clustering algorithms Expectation Maximization (EM) [122] and Self Organizing Maps [123] from WEKA tool, authors show that these algorithms are successfully able to classify attack samples using L3 cache misses counter.

$$g_k = \max\{0, g_{k-1} + log((d_{na}(k) + 1)/(d_n(k) + 1))\} >= h \quad (1)$$

CacheShield uses two HPCs: LLC misses (to decide if an attack is active) and total execution cycles (to decide if victim application is active or not). If cache misses is found to be greater than 0, it computes mean of cache misses samples (μ_a) and calculates a threshold h based on the calculated mean (threshold is used to make a detection decision). Then a detection rule g_k (given in (1)) is calculated for a sample k of cache misses. If g_k is greater than the threshold h, an alarm flag is raised. In (1), d_{na} is the distance between mean of the cache miss values for a cluster of no_attack samples and the cache miss value of the sample k. And d_a is the distance between mean of the cache miss values for a cluster of attack samples and the cache miss value of the sample k. For evaluation of *CacheShield*, few applications with high memory usage like Yahoo Cloud Serving Benchmark, Video Streaming, Randmem Benchmark are selected to create noisy conditions. Three crypto-algorithms of AES, ESA and ElGamal are used with three famous cache side channel attacks: Flush+Reload, Flush+Flush and Prime+Probe. Experimental results indicate that for all attacks *CacheShield* shows a detection rate of 100%. Moreover, the attacks against ElGamal are detected before 37% execution of the encryption in worst-case. For RSA, detection is achieved before 50% of the execution of decryption algorithm in worst case.

Another anomaly based CSCA detection solution has been proposed by Kulah *et al.* [114]. Kulah *et al.* [114] presented a semi-supervised method, *SpyDetector*, to detect cache-based SCAs at run time under variable load conditions. Detection mechanism is focused on the spy process which disputes on the shared resources used by the victim process. *SpyDetector* determines the shared resources which are used by the victim process, uses different useful features of HPC's to quantify between normal and abnormal contentions, correlates victim process associated with these resources or all the processes which are using the shared resource of interest and uses anomaly based machine learning approaches to detect abnormal level of contention. *SpyDetector* has been validated on CSCAs such as Prime+Probe on AES, Flush+Reload on AES & ECDSA and Flush+Flush on AES. Experiments revealed that *SpyDetector* can perform at run-time under variable load conditions in both physical and cross-VM configurations. Experimental evaluation shows that *SpyDetector* detects Prime+Probe attack with an average F-measure of 0.83, Flush+Reload with average F-measure of 0.99 for physical system and 0.97 for cross-VM setup and Flush+Flush with average F-measures of 0.82 for physical and 0.96 for cross-VM setups. The overall performance overhead for the system is between 0.49% to 3.58%.

3) ANOMALY + SIGNATURE-BASED DETECTION TECHNIQUES

As discussed earlier, some of the cache side channel detection techniques use a combination of both signature and anomaly based techniques. Examples of such techniques include: [75], [76], [86]. In the following, we discuss these techniques in detail.

Chiappetta *et al.* [86] proposed machine learning based detection of cache-based side-channel attacks. This work used three different approaches for detection of cache based side channel attacks. The first approach is based on correlation. If a correlation is found between a particular process and a victim process, it is an indication of an attack. The motivation behind this approach is that both victim and malicious processes act in similar ways (similar loops with similar operations). Experiments show that the number of last level cache accesses acts as a good parameter to detect an existing correlation. The second approach is based on Anomaly Detection. The particular method used is Gaussian Anomaly Detection [116]. In this work, authors build a model for malicious processes considering them normal and treat all other processes (normal) as anomalies. Authors state that the reason of doing this is that its practically impossible to build a model including all possible benign processes that can run on a system. The third approach is based on Neural Networks. Authors trained Neural Networks based on collected performance counters values (instruction and cache related events) for benign and malicious applications (responsible for attacking through side-channels). The machine learning methods (anomaly detection and neural network) rely on following hardware performance counters: executed instructions, total execution cycles, L2 cache hits, Last level cache (L3) accesses and misses. These events are selected based on experimental evidence. Chiappetta *et al.* [86] performed the assessment of Neural Network and Anomaly Detection using a metric of F-score [124]. The experimental results on an Intel Xeon machine while evaluating this proposed CSCA detection mechanism shows that the proposed technique can detect spy processes performing Flush+Reload [5] type of side-channel attacks with very high accuracy i.e. an F-score of 0.93 and 1.0 on AES and ECDSA crypto-systems by Neural Network. All three proposed methods are able to detect an attack in 1/5th of the time of attack completion.

Another technique which utilizes both signature and anomaly based detection has been proposed by Zhang *et al.* [75]. Zhang *et al.* [75] correlated execution of cryptographic application on a virtual machine (VM) with the anomalous behaviour of caches to detect cache side channel attacks in cloud systems. The proposed mechanism, *CloudRadar*, combines anomaly-based and signature-based attack detection techniques. Once an attack is detected, VM migration is performed as a countermeasure. *CloudRadar* serves as a lightweight patch to the cloud system under consideration. Zhang *et al.* [75] also identified that the two most important requirements for a signature to identify crypto application's execution are that it should be unique and should be repeatable. They used different types of events like CPU events, cache events and kernel software events to generate signature of applications. It is found that some events (like instructions, branches and mispredicted branch instructions, L1 instruction cache misses) are better for signature generation compared to others because of their uniqueness and ability to repeat. Their experiments showed that only a single feature of total number of branch operations out of the previously identified features was good enough to generate signatures and was used for further experiments. This work uses Dynamic Time warping (DTW) [125] algorithm to find distance between two sequences that represent signature and run-time measurements of performance counter values from untrusted VMs. When *CloudRadar* detects the execution of cryptographic application on the victim VM, the detection framework selects two short sub-sequences from the runtime sequence of performance counter values (cache misses and hits) being monitored on untrusted VMs. These sub-sequences correspond to ''data points of size w'' before and after the point of minimum DTW distance (DTW distance is used to detect the cryptographic application's execution and minimum DTW distance would correspond to the point when crypto application starts executing). If the difference between the values of the selected sub-sequences is found to be larger than a threshold, possibility of a side-channel attack is detected.

CloudRadar is evaluated using a system consisting of a controller server, a client server and two hosts cloud servers. Six different crypto applications belonging to category of symmetric and asymmetric crypto-systems are used for experiments (ElGamal and DSA from GnuPG, AES and 3DES from OpenSSL and hash: HMAC from OpenSSL and SHA512 from GnuPG). The proposed mechanism is tested using Prime+Probe and Flush+Reload cache based side channel attacks. *Cloud-Radar* is shown to have a 100% true positive rate (with no false positives) when performance counters are sampled at intervals of 100μs and DTW threshold is kept between 0.3 and 0.4. Sampling frequency of 1ms shows worse results while detecting execution of cryptographic applications. With a window size of w = 5 (w is discussed in previous paragraph), CloudRadar is able to achieve a false positive rate of 0% with a true positive rate of 100% at a sampling rate of 1ms. At lower values of w, the false positive rate is much higher (e.g. false positive rate is 20%-30% at w = 1). Detection latency/speed of CloudRadar is in the ''order of milliseconds'' on the used machine. Performance overhead of *CloudRadar* measured using a set of crypto applications, SPEC2006 [90] and CloudSuite [126] benchmarks is found to be little (within 5%).

A three-step detection method for cache and branch predictor based side channel attacks proposed by Alam *et al.* [76] also combines Anomaly and Signature based detection. The first step is used to detect the anomaly, the second step finds the class of anomaly (either related to branch or cache attacks) and the third step correlates malicious process with

the victim. This correlation is performed to reduce the number of false positives. At the first step of the method presented by Alam *et al.* [76], eight different performance events (branch instructions retired, branch instructions misses, Last Level Cache References, Last Level Cache Misses, Instruction Retired, UnHalted Core Cycles, UnHalted Reference Cycles, Bus Cycles) are monitored in parallel for a set of benign and malicious applications. The benign applications include commonly used Linux tools like *cd, gzip, mv etc.*. The data is then smoothed by using a finite impulse response filter known as *Simple Moving Average (SMA)* [127]. This filter calculates the ''unweighted mean of an equal number of data on either side of an intermediate value''. Next, the data is scaled with the help of *Standardization* [128] such that it achieves a mean of zero and a variance of one. Importance of features is then calculated using the technique of *Standard Stability Selection* [129]. This data is then used to train a semi-supervised anomaly detection mechanism known as One-Class Support Vector Machine (OC-SVM) [130], which is configured with a non-linear kernel (RBF). For the purpose of learning, this algorithm uses data with only one label. In other words, only the data belonging to one of the classes is labelled (also known as normal class). If any abnormality/anomaly is detected using this anomaly detector (OC-SVM), the process under consideration is passed to the already trained classifiers to determine the category of anomaly. The used classifiers in this work include: Random Forest [89], Adaboost [131], Multi-layer perceptron [132], Naive Bayes [133] and Support Vector Machine [134]. These classifiers are trained using the data from execution traces of different side-channel attacks that include different hardware events affected by those attacks. Finally to perform the third step of the proposed approach (correlation of the malicious process and the victim process), *Fast Dynamic Time Warping (fast-DTW)* [135] is used. If similarity between two temporal sequences composed of performance events is found to be above a threshold, the abnormal process would be detected as a side-channel attack. The proposed approach is validated experimentally with the help of cache side channel attacks on crypto-systems of AES [24] and Clefia [136] using two different hardware environments (Intel Core i5-4570 and Intel Xeon E5-2630 v3). The best accuracy for anomaly detection module is found to be 100% and 97% for both set-ups at a sampling granularity of 1ms. The best classification accuracy is shown by Adaboost classifier which is above 99% for both setups at sampling frequencies of 10 and 1 ms. The best accuracy of correlation module for setup 1 is found to be 83% at sampling frequency of 1ms with a DTW window of size $w = 5$ and for setup 2 it is 74% at sampling frequency of 1ms with a DTW window of size $w = 5$.

Possibility of side channel attacks can also be identified by detecting the presence of multiple VMs on same hardware in cloud systems as done by Zhang *et al.* [137] and Inci *et al.* [138]. In a public cloud, same physical machine may be shared by many VMs. Co-residency of different VMs on the same physical machine increases the risk of security breakdown. A VM with malicious intent can use the shared resources (like caches) on the same physical machine to attack a victim VM. Zhang *et al.* [137] proposed *HomeAlone* to detect cross-VM side channel attacks by first detecting the existence of untrusted VM on the same physical server. *HomeAlone* is implemented at the level of hypervisor/VM. It works by observing the cache memory activity on the victim VM. *HomeAlone* works in three steps: first step (PRIME) fills up a portion of the shared cache by reading data from main memory. In the second step (IDLE), it waits for a specific amount of time while other VMs are running. In the third step (PROBE), *HomeAlone* reads the same cache section and uses time of the reading to determine if this portion is overwritten by another VM. Any time difference indicates the presence of shared resources and possibility of side channel attacks. Experimental evaluation of *HomeAlone* is performed using an adversary VM running Prime+Probe attack. The evaluation shows that the detection accuracy is improved with the increase in frequency of Prime+Probe attack or with increased cache sets monitored by *HomeAlone* which overlap with malicious VM's activity region. A true detection rate of 85% is observed when 1/16th of cache scanned by *HomeAlone* overlaps with malicious VM's activity region. Further, the maximum performance overhead (using PARSEC benchmarks [91]) is found to be equal to 4.6% (with most of the cases around 2%).

Inci *et al.* [138] also focused on the problem of detecting co-location required to perform cross-VM attacks such as Prime+Probe and Flush+Reload in enterprise clouds. This work demonstrates three co-location detection methods named as; cooperative last-level cache covert channel, software profiling on LLC and memory bus locking. Co-location problem is analyzed on threat models of Amazon EC2 Cloud, Google Cloud Engine and Microsoft Azure. Contribution of this paper includes; devising a new LLC software profiling tool which is able to detect application by non-collaborating co-located victims in cloud, this tool is able to detect without the help of memory de-duplication and any other sharing mechanism and describing three co-location methods and discussing their success on popular clouds (considered as a threat model). Threat model considers two attack scenarios for cross-VM on public clouds i.e., the target victim is predefined or the target victim is unknown. Targeted colocation includes identification information of the victim e.g. IP address. Attacker reforms instances on the cloud until the targeted victim is co-located on the same physical machine. Using the IP, attacker can check the server which is creating CPU load and then co-location tests can be run to verify the presence of victim. It is very easy to achieve co-location detection in this case but one needs to run many tests on the same physical machine as of victim.

For random victim co-location detection, attacker sends instances on cloud until it is confirmed that instance is not alone e.g. is co-located with any other VM. The goal is to get maximum likelihood and reduction in the cost of co-locating with viable target. Less costly instances use less CPU cores

which tend to share same hardware at maximum. That is why such instances have bright chance of co-location. Results explain that collaborative and non-collaborative co-location to certain clouds is possible on major cloud services. Proposed mechanism was able to achieve targeted co-location in Amazon EC2 with the help of LLC software profiling (for RSA and AES crypto-systems). For memory bus locking mechanism, memory accesses lead to major degradation while in covert channel, the method achieves high accuracy. It is demonstrated in the paper that LLC software profiling mechanism can be used for co-location detection without use of memory de-duplication and any other sort of sharing from victim side. There exist other techniques as well [139]–[142] to monitor executing guest VMs which can be used for detection of co-residency and eventually side-channels.

As discussed in Section 2.4, Younis *et al.* [63] surveyed and compared two CSCA mitigation techniques (cache flushing [143] and noise injection [144]) and two CSCA detection techniques (*HomeAlone* [137] and a two stage detection technique proposed by Yu *et al.* [64]). These CSCA detection techniques have already been discussed in this section. Younis *et al.* [63], on comparing these CSCA detection and prevention mechanisms, found out that Flushing technique was able to mitigate all the three attacks but injecting noise was unable to detect Prime+Probe and Flush+Reload (4-10 times out of 20) which reduces their detection accuracy to half. For preventing context-switching, cache flushing also induces a high affect on cache efficiency. It is discussed that all prevention and detection mechanisms affect the cache usefulness e.g. solution proposed by Yu *et al.* [64] slows the CPU operations to count cache misses which significantly reduces the effectiveness of cache whereas, *HomeAlone* solution flushes the data every time and forces CPU cache to write it back from main memory which degrades the effectiveness of cache. The paper further observes that flushing and injecting noise can prevent cache at all levels. Two stage detection solution [64] can detect CSCAs at all levels, whereas *HomeAlone* detects attack at only L2 cache level.

V. HIGH-LEVEL SYNTHESIS

This section will discuss the summary of our findings based on the study of CSCA detection techniques found in literature. Table 2 summarizes the discussion on reviewed CSCA detection techniques. This table shows the extent to which proposed CSCA detection mechanisms have been evaluated by the studied papers. Major findings from the surveyed literature related to CSCA detection mechanisms are following:

- Cache Side Channel Attack (CSCA) detection techniques are largely divided into Signature Based and Anomaly Based detection techniques.
- Most of the CSCA detection techniques are Signature based techniques as shown in Table 2. There also exist few research works that use a combination of Anomaly and Signature based detection techniques. As mentioned above, most of the CSCA detection techniques are actually signature based detection techniques. However, generally in the domain of information security, signature based detection techniques have shown to be evaded with the help of obfuscation [145]. We note that the similar issue exists in the detection of cache side channel attacks. For example, assume a detection technique which uses a signature based on the HPCs (cache events, and branch events), such that an attack is represented by high values of cache events but low values of branch events. If an attacker is aware of the way the detection mechanism works, the attacker can use obfuscation by embedding some code in the attack process which will lead to high branch events as well, thus decreasing the probability of attack getting detected by the employed detection mechanism. We have not found any papers which target this issue and propose obfuscation proof detection mechanisms.

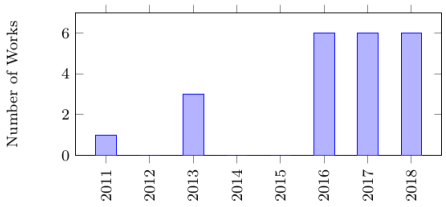


FIGURE 9. Number of CSCA detection research works.

- As shown in Figure 9, more than 80% of the research works focusing on detection of cache based side channel attacks are performed in last 3 years indicating that the field still lacks maturity.
- Almost all of the reviewed detection techniques use hardware performance counters available on all modern processors. A few works [75], [83], [107], [138] don't use HPCs but still rely on hardware timers provided by the processor vendor.
- Machine learning is also used notably for CSCA detection as 50% of the studied techniques apply machine learning models to recognize cache side channel attacks.
- It is generally believed that anomaly based detection techniques are capable of detecting unknown or zero-day attacks [75]. However, none of the surveyed research works has shown any empirical evidence of the capability of detecting unknown or modified attacks.
- We found that almost all of the CSCA detection solutions are purely software-based and there is only a single work [146] that also proposed a hardware implementation of the proposed detection technique.
- We also observed that there are many research papers that missed one or more important evaluation parameters while evaluating their CSCA detection proposals as shown in Table 2.

VI. TRENDS IN CSCA DETECTION: THE CHALLENGES, PITFALLS AND PERILS

In this section, we discuss the general trends in cache based side channel attack detection and the associated challenges

TABLE 3. List of security papers using HPCs in SCAs.

No.	Authors	HPCs' Pitfalls Addressed	HPCs' Pitfalls Resolved	HPCs Recommended
1	Martin et al. [57]	No	No	Yes
2	Uhsadel et al. [150]	No	No	Yes
3	Bhattacharya & Mukhopadhyay [151]	No	No	Yes
4	Chiapetta et al. [86]	No	No	Yes
5	Maurice et al. [102]	No	No	Yes
6	Payer [153]	No	No	Yes
7	Zhang et al. [75]	No	No	Yes
8	Nomani and Szefer [154]	Yes (Non determinism)	Yes	Yes
9	Gulmezoglu [155]	No	No	Yes
10	Irazoki [156]	Yes (Non determinism)	Yes	Yes
11	Allaf et al. [78]	No	No	Yes
12	Mushtaq et al. [85], [97], [98], [99]	No	No	Yes

which need to be considered when devising CSCA detection techniques.

A. USE OF HPCs IN SECURITY

Hardware performance monitoring counters have been significantly used in security attack detection mechanisms. Therefore, it is critical to understand the limitations [149] (discussed in next subsection) of these counters in order to appropriately use them in detection solutions. To lessen the burden on programmers, a number of utilities and tools are available to obtain counter information on variable platforms.

We analyzed the papers which use HPCs (hardware performance monitoring counters) for detection of CSCAs as shown in Table 3. Table 3 also lists the papers which acknowledged and addressed the problems while sampling HPCs.

B. ISSUES AND LIMITATIONS OF HPCs

The issues and limitations associated with the use of HPCs are following:

1) NON-DETERMINISM

Hardware performance counters may produce deterministic results when run in a strictly controlled environment [157]. Deterministic results of hardware performance counters also depend on the tools which you use for measurement of results. Generally, the values of hardware performance counters can vary from 1-10% (during different runs) due to non-determinism [157]. Only few hardware performance counters can produce deterministic results like retired instruction. Many of the other counters which measure events like cache and cycle counts are not deterministic on modern out-of-order machines due to various reasons which include: operating system activity, context switching, hardware interrupts, cost of measuring hardware performance counters, variations in tools for measurement [157]. Therefore, to use hardware performance counters for security applications, it is important to keep this non-determinism in mind and potentially use such events only in applications where this non-determinism can be tolerated (or potentially rely only on deterministic events).

2) MULTIPLEXING ISSUES

Multiplexing allows more counters to be used simultaneously than are physically supported by the hardware. With multiplexing, the physical counters are time-sliced, and the counts are estimated from the measurements. Naive use of multiplexing could lead to erroneous results that would not be detected by the user. Erroneous results can occur when the run-time is insufficient to permit the estimated counter values to converge to their expected values [158]. Reference [159] also study the accuracy of performance counter-based measurements. However, their focus is on the accuracy of measurements when the number of events to measure is greater than the number of the available performance counter registers. They compare two "time interpolation" approaches, multiplexing and trace alignment, and evaluate their accuracy. Their work does not address the measurement error caused by any software infrastructure that reads out and virtualizes counter values. Thus, heavy multiplexing of hardware performance counters should be avoided to prevent incorrect counter values that will be eventually used by attack detection techniques.

3) PERFORMANCE OVERHEAD

The use of hardware performance counters also introduces a performance overhead (incurred during the start/stop of counters and whenever they are read using some programming interface). The interfaces (used to control and read counters) introduce an overhead in the form of extra instructions, system calls, and can cause cache pollution that can change the cache and memory behavior of the monitored application [158]. The cost of processing counter overflow interrupts can be a significant source of overhead in sampling-based profiling. The performance cost of using these counters makes it necessary to only use these counters when necessary if real time CSCA detection solutions are desired.

C. USE OF ML IN SECURITY

In the recent past, ML has been extensively used in security domain e.g. malware and intrusion detection: [160]–[167].

Similarly, ML is employed in detection of CSCAs: [76]–[80], [85]–[89], [97], [99], [112]. Next, there is a discussion on the issues associated to the use of ML in CSCA detection.

D. ISSUES AND LIMITATIONS OF ML

As discussed previously, CSCA detection is a binary classification problem and ML offers many classifiers for the problem of detection. However, as a designer of CSCA detection mechanism, it is important to understand the rationale behind applying a specific ML algorithm for a particular detection problem. For example, a fundamental question should always be asked that if ML will be on overkill for the problem at hand. The usability of a ML model is generally judged based on two parameters; the model should achieve high accuracy and provide less implementation complexity which renders less performance cost. As we know CSCAs are very stealth in nature and take microseconds to execute, the detection mechanism using ML classifiers should be able to perform early stage detection (before the completion of attack) followed by mitigation mechanism to act before the attack retrieves the victim's confidential information. To assess the accuracy of a ML classifier, it is also critical to use proper metrics (examples include percentage accuracy, F-score) The adaptability and scalability of a detection mechanism highly depends on its run-time performance overhead. It is important that ML model has low implementation cost and can be easily embedded in the detection module. Models which explode in tree based nature, or perform a lot of forward and backward tracing, do conditional statements, take into account a lot of training data points, cost highly in terms of implementation complexity (although they can be very accurate in decision making). Therefore, it is crucial to focus on the balance in terms of detection accuracy and performance overhead before relying on a ML model for CSCA detection purposes.

VII. CONCLUSION AND FUTURE OUTLOOK

This work is the first effort to perform a detailed survey of CSCA detection techniques proposed in the last decade or so. We have identified a broad set of criterion to characterize the research works on CSCA detection. We discuss the detailed working of CSCA detection methodologies found in literature and provide a comparative summary of these techniques. We found that the research efforts in this domain are on the rise and there is a space to improve the existing techniques using innovative ideas. We provide a brief discussion on the future research directions that can be explored in the field of CSCA detection.

We believe that a lot of concepts from the field of malware and intrusion detection can be borrowed to solve the problem of CSCA detection. The field of malware detection seems to have more maturity, therefore, a lot of research ideas [168]–[171] can be adopted for the case of CSCA detection. We observed that almost 50% of the reviewed research works utilize machine learning classifiers to detect CSCAs. Most of these works use multiple machine learning models. Therefore, it would be interesting to explore the use of ensemble learning techniques to combine various classifiers and observe the impact on the overall detection results. As discussed in the previous section, almost all of the proposed CSCA detection solutions work entirely in the software. We believe an important future direction would be to explore possibility of hardware implementation of the proposed solutions or to think of hardware solutions from scratch. These hardware solutions can accelerate the response of detection solutions in the presence of an attack. Moreover, hardware based solutions can lead to faster mitigation solutions as well without involvement of software or OS. For example in case a hardware detector detects an attack, the processor pipeline can be sent a signal to stall instantaneously (without any lag due to software involvement) to make sure that no critical information is lost. Possible choices for implementation of hardware solutions include: separate cores or other programmable logic devices/systems (SoCs, NoCs etc.).

As there are various techniques for CSCAs and new techniques keep on appearing, CSCA detection solutions need to be more generic and adaptable. What we have observed from our study of the already published research is that the proposed techniques usually use a limited set of attacks to validate their proposals. Moreover, often the proposed machine learning classifiers are attack specific. We believe that in future the community needs to come up with more inclusive solutions and validate them on a wider variety of attacks. Similarly, the need to build detection techniques that would work for zero-day, unknown or modified attacks is evident. We discussed the need for detection based CSCA mitigation solutions in the earlier part of this paper. However, we have not seen research works that integrate the two. It is important to experiment with such ideas as their integration would expose new challenges that we have not been able to observe before.

As Machine Learning has been able to help CSCA detection techniques significantly, there are other areas that can be applied to solve CSCA detection problem. These areas include Deep Learning [172], Game Theory [173] and Fuzzy Logic [174], [160]. These areas have already been extensively applied to solve the problem of malware and intrusion detection: [160]–[167].

We have also observed that all of the proposed CSCA detection techniques focus on Intel'x x86 architecture. However, attacks on other architectures like ARM have also been proposed [21]–[23]. Now, since characteristics of attacks on different architectures can be different, the challenges to detect such attacks can be different on different architectures. Therefore, it would be worthwile to study detection of CSCAs on other architectures like ARM. Similarly attacks on ARM's support for isolated execution environment i.e ARM TrustZone have been shown to be possible [175], [176]. However, a detailed study of such attacks and demonstration of their detection is still missing. We noted that the studied proposed detection techniques focus on detection of cache side channel attacks on cryptographic execution

(e.g. RSA, AES, ECDSA). However, cache side channel attacks exist on other targets as well like user and kernel space ASLR (Address space layout randomization) [177] and other environments like browsers and non-native code (e.g. javascript) [21], [178]. In future researchers will have to come up with detection techniques for attacks against such targets.

There exist research works [179]–[181] that have proposed techniques to detect side channel vulnerabilities using program analysis. Combining such techniques with CSCA detection methods can help to reduce burden on CSCA detection and increase the confidence of detection as well. Moreover, compiler assistance can prove to be useful in this regard as well. Such solutions would also help to reduce performance overheads of run-time detection. It is clear that at the moment there exist quite a few challenges in this field of CSCA detection techniques and there is a need to invest more resources and minds in this domain to solve these critical problems.

ACKNOWLEDGMENT

This work was supported in part by the PHC PERIDOT Project e-health SECURE under Grant 3-6/HEC/R&D/PERIDOT/2017. The authors would also like to thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] (2017). *IBM Research*. Accessed: Oct. 10, 2017. [Online]. Available: https://www.ibm.com/

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2009, pp. 199–212.

[3] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud," in *Proc. 21st USENIX Conf. Secur. Symp.*, Berkeley, CA, USA, 2012, p. 9.

[4] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An exploration of L2 cache covert channels in virtualized environments," in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop (CCSW)*, New York, NY, USA, 2011, pp. 29–40.

[5] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, L3 cache side-channel attack," in *Proc. 23rd USENIX Conf. Secur. Symp. (SEC)*, Berkeley, CA, USA, 2014, pp. 719–732.

[6] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2012, pp. 305–316.

[7] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Upper Saddle River, NJ, USA: Pearson, 2017.

[8] Y. Yarom, D. Genkin, and N. Heninger, *CacheBleed: A Timing Attack on OpenSSL Constant Time RSA*. Berlin, Germany: Springer, 2016, pp. 346–367.

[9] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. 19th Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, London, U.K.: Springer-Verlag, 1999, pp. 388–397.

[10] J.-J. Quisquater and D. Samyde, *ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards*. Berlin, Germany: Springer, 2001, pp. 200–210.

[11] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games—Bringing access-based cache attacks on AES to practice," in *Proc. IEEE Symp. Secur. Privacy*, Washington, DC, USA, May 2011, pp. 490–505.

[12] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures," *J. Cryptol.*, vol. 23, no. 1, pp. 37–71, Jan. 2010.

[13] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *J. Cryptograph. Eng.*, vol. 8, no. 1, pp. 1–27, Apr. 2018.

[14] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proc. 13th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, vol. 9721. New York, NY, USA: Springer-Verlag, 2016, pp. 279–299.

[15] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *Proc. 24th USENIX Conf. Secur. Symp.*, Berkeley, CA, USA, 2015, pp. 897–912.

[16] D. A. Osvik, A. Shamir, and E. Tromer, *Cache Attacks Countermeasures: The Case AES*. Berlin, Germany: Springer, 2006, pp. 1–20.

[17] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in PaaS clouds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2014, pp. 990–1003.

[18] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, *Wait a Minute! A Fast, Cross-VM Attack on AES*. Cham, Switzerland: Springer, 2014, pp. 299–319.

[19] Y. Zhang, "Cache side channels: State of the art and research opportunities," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 2617–2619.

[20] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, Washington, DC, USA, May 2015, pp. 605–622.

[21] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on ARM and their implications for Android devices," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 858–870.

[22] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *Proc. 25th USENIX Secur. Symp.*, Austin, TX, USA, 2016, pp. 549–564.

[23] M. Green, L. Rodrigues-Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, "Autolock: Why cache attacks on arm are harder than you think," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1075–1091.

[24] Z. He and R. B. Lee, "How secure is your cache against side-channel attacks?" in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 341–353.

[25] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," Dept. Comput. Sci., Univ. Bristol, Bristol, U.K., Tech. Rep. CSTR-02-003, 2002.

[26] D. J. Bernstein, "Cache-timing attacks on AES," Dept. Math., Statist., Comput. Sci., Univ. Illinois Chicago, Chicago IL, USA, Tech. Rep., 2005.

[27] O. Aciiçmez, W. Schindler, and C. K. Koç, "Cache based remote timing attack on the AES," in *Proc. 7th Cryptograph. Track RSA Conf. Topics Cryptol. (CT-RSA)*. Berlin, Germany: Springer-Verlag, 2006, pp. 271–286.

[28] T. Tsunoo, Y. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, *Cryptanalysis of DES Implemented on Computers With Cache*. Berlin, Germany: Springer, 2003, pp. 62–76.

[29] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Proc. 8th Int. Conf. Cryptograph. Hardw. Embedded Syst. (CHES)*. Berlin, Germany: Springer-Verlag, 2006, pp. 201–215.

[30] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *IACR Cryptol. ePrint Arch.*, vol. 8, p. 613, Apr. 2016.

[31] T. Kim, M. Peinado, and G. Mainar-Ruiz, "Stealthmem: System-level protection against cache-based side channel attacks in the cloud," in *Proc. 21st USENIX Conf. Secur. Symp.*, Berkeley, CA, USA, 2012, p. 11.

[32] O. Aciicmez and C. K. Koc, "Trace-driven cache attacks on AES (short paper)," in *Proc. 8th Int. Conf. Inf. Commun. Secur. (ICICS)*. Berlin, Germany: Springer-Verlag, 2006, pp. 112–121.

[33] J.-F. Gallais, I. Kizhvatov, and M. Tunstall, "Improved trace-driven cache-collision attacks against embedded AES implementations," in *Proc. 11th Int. Conf. Inf. Secur. Appl. (WISA)*. Berlin, Germany: Springer-Verlag, 2011, pp. 243–257.

[34] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A timing attack on OpenSSL constant-time RSA," *J. Cryptograph. Eng.*, vol. 7, no. 2, pp. 99–112, Jun. 2017.

[35] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Proc. Cryptograph. Track at RSA Conf. Topics Cryptol. (CT-RSA)*. Berlin, Germany: Springer-Verlag, 2006, pp. 1–20.

[36] C. Percival, "Cache missing for fun and profit," in *Proc. BSDCan*, 2005, pp. 1–13.

[37] O. Aciiçmez, "Yet another micro architectural attack: Exploiting I-cache," in *Proc. ACM Workshop Comput. Secur. Archit. (CSAW)*, New York, NY, USA, 2007, pp. 11–18.

[38] O. Aciicmez, B. B. Brumley, and P. Grabher, "New results on instruction cache attacks," in *Proc. 12th Int. Conf. Cryptograph. Hardw. Embedded Syst. (CHES)*, Berlin, Germany: Springer-Verlag, 2010, pp. 110–124.

[39] B. B. Brumley and R. M. Hakala, *Cache-Timing Template Attacks*. Berlin, Germany: Springer, 2009, pp. 667–684.

[40] O. Aciicmez and W. Schindler, "A vulnerability in rsa implementations due to instruction cache analysis and its demonstration on openssl," in *Proc. Cryptopgraph. Track RSA Conf. Topics Cryptol. (CT-RSA)*. Berlin, Germany: Springer-Verlag, 2008, pp. 256–273.

[41] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop (CCSW)*, New York, NY, USA, 2011, pp. 41–46.

[42] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space ASLR," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2013, pp. 191–205.

[43] J. van de Pol, N. Smart, and Y. Yarom, "Just a little bit more," in *Topics in Cryptology* (Lecture Notes in Computer Science), vol. 9048. New York, NY, USA: Springer, 2015, pp. 3–21.

[44] T. Allan, B. B. Brumley, K. Falkner, J. van de Pol, and Y. Yarom, "Amplifying side channels through performance degradation," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl. (ACSAC)*, New York, NY, USA, 2016, pp. 422–435.

[45] N. Benger, J. Pol, N. P. Smart, and Y. Yarom, "'Ooh aah... just a little bit' : A small amount of side channel can go a long way," in *Proc. 16th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, vol. 8731. New York, NY, USA: Springer-Verlag, 2014, pp. 75–92.

[46] C. P. García, B. B. Brumley, and Y. Yarom, "Make sure DSA signing exponentiations really are constant-time," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)* New York, NY, USA, 2016, pp. 1639–1650.

[47] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, p. 35, 2012.

[48] D. Page, "Partitioned cache architecture as a side-channel defence mechanism," Dept. Comput. Sci., IACR Cryptol. ePrint Arch., Univ. Bristol, Bristol U.K., Tech. Rep. 2005/280, Jun. 2005, p. 280.

[49] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proc. 34th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2007, pp. 494–505.

[50] 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 *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 406–418.

[51] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 494–505, Jun. 2007.

[52] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 393–404.

[53] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *Proc. 41st IEEE/ACM Int. Symp. Microarchitecture*, Nov. 2008, pp. 83–93.

[54] H. M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip," in *ACM SIGARCH Comput. Archit. News*, vol. 41, pp. 583–594, Jun. 2013.

[55] W.-M. Hu, "Reducing timing channels with fuzzy time," *J. Comput. Secur.*, vol. 1, nos. 3–4, pp. 233–254, Oct. 1992.

[56] J. W. Gray, "On introducing noise into the bus-contention channel," in *Proc. IEEE Comput. Soc. Symp. Res. Secur. Privacy*, May 1993, pp. 90–98.

[57] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 118–129, 2012.

[58] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 479, Sep. 2016.

[59] S. Anwar, Z. Inayat, M. F. Zolkipli, J. M. Zain, A. Gani, N. B. Anuar, M. K. Khan, and V. Chang, "Cross-VM cache-based side channel attacks and proposed prevention mechanisms: A survey," *J. Netw. Comput. Appl.*, vol. 93, pp. 259–279, Sep. 2017.

[60] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," *J. Hardw. Syst. Secur.*, vol. 2, no. 1, pp. 33–50, Mar. 2018.

[61] Y. Jin, "Introduction to hardware security," *Electron.*, vol. 4, no. 4, pp. 763–784, 2015.

[62] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," *IACR Cryptol. ePrint Arch.*, vol. 2005, p. 388, Sep. 2005.

[63] Y. A. Younis, K. Kifayat, and A. Hussain, "Preventing and detecting cache side-channel attacks in cloud computing," in *Proc. 2nd Int. Conf. Internet Things, Data Cloud Comput. (ICC)*, New York, NY, USA, 2017, pp. 83:1–83:8.

[64] S. Yu, X. Gui, and J. Lin, "An approach with two-stage mode to detect cache-based side channel attacks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2013, pp. 186–191.

[65] *Intel 64 and ia-32 Architectures Developer's Manual*, Intel, Santa Clara, CA, USA, 2013.

[66] (2017). *PerfMon*. Accessed: Oct. 10, 2017. [Online]. Available: https://knowledge.ni.com/

[67] (2017). *OProfile*. Accessed: Oct. 10, 2017. [Online]. Available: http://oprofile.sourceforge.net/

[68] (2015). *PERF: Linux Profiling With Performance Counters*. Accessed: Oct. 10, 2017. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[69] *Intelâő VTuneâđć Profiler Performance Analysis Cookbook*. Accessed: Oct. 10, 2017. [Online]. Available: https://software.intel.com/en-us/vtune-amplifier-cookbook, 2017.

[70] (2017). *Performance Application Programming Interface*. Accessed: Oct. 10, 2017. [Online]. Available: http://icl.cs.utk.edu/papi/

[71] M. Stamp, *Introduction to Machine Learning With Applications in Information Security*. London, U.K.: Chapman & Hall, 2017.

[72] M. B. Christopher, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, 2016.

[73] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1. New York, NY, USA: Springer, 2001.

[74] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112. New York, NY, USA: Springer, 2013.

[75] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. New York, NY, USA: Springer, 2016, pp. 118–140.

[76] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks," Cryptol. ePrint Arch., Indian Inst. Technol., Kharagpur, India, Tech. Rep. 2017/564, 2017. [Online]. Available: https://eprint.iacr.org/2017/564

[77] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, Jul. 2013.

[78] Z. Allaf, M. Adda, and A. Gegov, "A comparison study on flush+reload and prime+probe attacks on AES using machine learning approachess," in *Proc. UK Workshop Comput. Intell.*, 2017, pp. 203–213.

[79] Z. Allaf, M. Adda, and A. Gegov, "ConfMVM: A hardware-assisted model to confine malicious VMs," in *Proc. UKSim-AMSS 20th Int. Conf. Comput. Model. Simulation (UKSim)*, Mar. 2018, pp. 49–54.

[80] M. Payer, "HexPADS: A platform to detect 'stealth' attacks," in *Proc. Int. Symp. Eng. Secure Softw. Syst.* New York, NY, USA: Springer, 2016, pp. 138–154.

[81] S.-H. Peng, Q.-F. Zhou, and J.-L. Zhao, "Detection of cache-based side channel attack based on performance counters," in *Proc. 3rd Int. Conf. Artif. Intell. Ind. Eng. (AIIE)*, 2017, pp. 377–381.

[82] S. Briongos, P. Malagón, J. L. Risco-Martín, and J. M. Moya, "Modeling side-channel cache attacks on AES," in *Proc. Summer Comput. Simulation Conf.*, 2016, p. 37.

[83] M. Chouhan and H. Hasbullah, "Adaptive detection technique for cache-based side channel attack using Bloom filter for secure cloud," in *Proc. 3rd Int. Conf. Comput. Inf. Sci. (ICCOINS)*, Aug. 2016, pp. 293–297.

[84] A. Raj and J. Dharanipragada, "Keep the PokerFace on! Thwarting cache side channel attacks by memory bus monitoring and cache obfuscation," *J. Cloud Comput.*, vol. 6, no. 1, p. 28, Dec. 2017.

[85] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "NIGHTs-WATCH: A cache-based side-channel intrusion detector using hardware performance counters," in *Proc. 7th Int. Workshop Hardw. Architectural Support Secur. Privacy (HASP)*, 2018. p. 1

[86] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, vol. 49, pp. 1162–1174, Dec. 2016.
[87] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.
[88] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[89] R. Lippmann, "An introduction to computing with neural nets," *IEEE Assp Mag.*, vol. 4, no. 2, pp. 4–22, 1987.
[90] (2017). *SPEC CPUâő 2006*. Accessed: Oct. 10, 2017. [Online]. Available: https://www.spec.org/cpu2006/
[91] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2008, pp. 72–81.
[92] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: A side-channel attack detection tool for tracking prime+probe," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.
[93] I. T. Jolliffe, "Principal components in regression analysis," in *Proc. Principal Compon. Anal.*, 2002, pp. 167–198.
[94] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, and Q. V. Le, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.
[95] J. R. Quinlan, *Programs for Machine Learning*. Amsterdam, The Netherlands: Elsevier, 2014.
[96] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! A fast, cross-VM attack on AES," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. New York, NY, USA: Springer, 2014, pp. 299–319.
[97] M. Mushtaq, A. Akram, M. K. Bhatti, R. N. B. Rais, V. Lapotre, and G. Gogniat, "Run-time detection of prime + probe side-channel attack on AES encryption algorithm," in *Proc. Global Inf. Infrastruct. Netw. Symp. (GIIS)*, Oct. 2018, pp. 1–5.
[98] M. Mushtaq, A. Akram, M. Khurram Bhatti, U. Ali, V. Lapotre, and G. Gogniat, "Sherlock Holmes of cache side-channel attacks in Intel's x86 architecture," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Washington, DC, USA, Jun. 2019, pp. 1–9.
[99] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, "Machine learning for security: The case of side-channel attack detection at run-time," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 485–488.
[100] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," in *Proc. Black Hat*, 2015, pp. 7–9.
[101] A. Barresi, K. Razavi, M. Payer, and T. R. Gross, "Cain: Silently breaking ASLR in the cloud," in *Proc. WOOT*, vol. 15, 2015, p. 45.
[102] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: Cross-cores cache covert channel," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. New York, NY, USA: Springer, 2015, pp. 46–64.
[103] D. Carraway. (2013). *lookbusy—A Synthetic Load Generator*. Accessed: Oct. 10, 2017. [Online]. Available: http://www.devin.com/lookbusy
[104] R. Longbottom. (2016). *Roy Longbottom's PC Benchmark Collection*. [Online]. Available: http://www.roylongbottom.org.uk/
[105] J. D. McCalpin. (1995). *STREAM benchmark*. Accessed: Oct. 10, 2017. [Online]. Available: https://www.cs.virginia.edu/stream/
[106] (2004). *SysBench: A System Performance Benchmark*. Accessed: Oct. 10, 2017. [Online]. Available: http://sysbench.sourceforge.net/
[107] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with Déjávu," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 7–18.
[108] (2018). *Nbench-Byte Benchmark*. Accessed: Oct. 10, 2018. [Online]. Available: http://www.math.cmu.edu/ florin/bench-32-64/nbench/
[109] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[110] A. W. Paundu, D. Fall, D. Miyamoto, and Y. Kadobayashi, "Leveraging KVM events to detect cache-based side channel attacks in a virtualization environment," *Secur. Commun. Netw.*, vol. 2018, Feb. 2018, Art. no. 4216240.
[111] S. Rostedt. (2014). *Ftrace Kernel Hooks, More Than Just Tracing*. Accessed: Oct. 10, 2017. [Online]. Available: https://blog.linuxplumbersconf.org/2014/ocw/sessions/1773
[112] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in *Proc. 3rd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Apr. 2018, pp. 7–12.
[113] S. Briongos, G. Irazoqui, P. Malagón, and T. Eisenbarth, "CacheShield: Detecting cache attacks through self-observation," in *Proc. 8th ACM Conf. Data Appl. Secur. Privacy*, 2018, pp. 224–235.
[114] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "SpyDetector: An approach for detecting side-channel attacks at runtime," *Int. J. Inf. Secur.*, vol. 18, no. 4, pp. 393–422, Aug. 2019.
[115] (2018). *Benefits of Intel Cache Monitoring Technology in the Intel Xeon Processor E5 C3 Family*. [Online]. Available: https://software.intel.com/en-us/blogs/2014/06/18/benefit-of-cache-monitoring
[116] A. Kristiadi. (2018). *Gaussian Anomaly Detection*. Accessed: Oct. 10, 2018. [Online]. Available: https://wiseodd.github.io/techblog/2016/01/16/gaussian-anomaly-detection/
[117] (2018). *Flush+Flush Side-Channel Attack Github Repository*. Accessed: Oct. 10, 2018. [Online]. Available: https://github.com/iaik/flush_flush
[118] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, nos. 1–2, pp. 100–115, Jun. 1954.
[119] M. Basseville and I. V. Nikiforov, *Detection Abrupt Changes: Theory Application*, vol. 104. Englewood Cliffs, NJ, USA: Prentice & Hall, 1993.
[120] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[121] K. Kira and L. A. Rendell, "The feature selection problem: Traditional methods and a new algorithm," in *Proc. AAAI*, vol. 2, 1992, pp. 129–134.
[122] X. Jin and J. Han, "Expectation maximization clustering," in *Encyclopedia of Machine Learning*. New York, NY, USA: Springer, 2011, pp. 382–383.
[123] S.-O. Map and T. Kohonen, "Self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.
[124] B. C. Vickery, "Reviews: Van rijsbergen, cj information retrieval. 2nd ed. London, butterworths, i978. 208pp," *J. Librarianship*, vol. 11, no. 3, p. 237, Jul. 1979.
[125] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," in *Readings speech Recognition*. Amsterdam, The Netherlands: Elsevier, 1990, pp. 159–165.
[126] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *ACM SIGPLAN Notices*, vol. 47, pp. 37–48, ACM, 2012.
[127] Y. lun Chou, "Statistical analysis," *Holt International*. New York, NY, USA: Holt, Rinehart and Winston, 1975.
[128] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed. New York, NY, USA: Academic, 2008.
[129] N. Meinshausen and P. Bühlmann, "Stability selection," *J. Roy. Stat. Soc., B Stat. Methodol.*, vol. 72, no. 4, pp. 417–473, 2010.
[130] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001.
[131] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[132] H. D. Block, "The perceptron: A model for brain Functioning. I," *Rev. Modern Phys.*, vol. 34, no. 1, pp. 123–135, Jan. 1962.
[133] K. P. Murphy, "Naive Bayes classifiers," Univ. Brit. Columbia, Vancouver, BC, Canada, Tech. Rep., 2006, vol. 18.
[134] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, Jul./Aug. 2008.
[135] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, Oct. 2007.
[136] C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga, "Cache timing attacks on clefia," in *Proc. Int. Conf. Cryptol.* New Delhi, India: Springer, 2009, pp. 104–118.
[137] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "HomeAlone: Co-residency detection in the cloud via side-channel analysis," in *Proc. IEEE Symp. Secur. Privacy*, May 2011, pp. 313–328.
[138] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.*, Santa Barbara, CA, USA, vol. 9813, 2016, pp. 368–388.
[139] D. J. Dean, H. Nguyen, and X. Gu, "UBL: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proc. 9th Int. Conf. Autonomic Comput. (ICAC)*, 2012, pp. 191–200.

[140] F. Doelitzscher, M. Knahl, C. Reich, and N. Clarke, "Anomaly detection in IaaS clouds," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2013, pp. 387–394.

[141] B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. GT-CS-11-05, 2011.

[142] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in linux containers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2015, pp. 1–5.

[143] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," *IEEE Trans. Cloud Comput.*, vol. 2, no. 4, pp. 395–408, Oct. 2014.

[144] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2013, pp. 827–838.

[145] G. Canfora, A. Di Sorbo, F. Mercaldo, and C. A. Visaggio, "Obfuscation techniques against signature-based detection: A case study," in *Proc. Mobile Syst. Technol. Workshop (MST)*, May 2015, pp. 21–26.

[146] J. Demme and S. Sethumadhavan, "Rapid identification of architectural bottlenecks via precise event counting," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2011, pp. 353–364.

[147] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim., Feedback-Directed Runtime Optim.*, 2004, p. 75.

[148] Z. Wang, S. Peng, X. Guo, and W. Jiang, "Zero in and timefuzz: Detection and mitigation of cache side-channel attacks," in *Proc. Int. Conf. Secur. Inf. Technol. Commun.* New York, NY, USA: Springer, 2018, pp. 410–424.

[149] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "SoK: The challenges, pitfalls, and perils of using hardware performance counters for security," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 20–38.

[150] L. Uhsadel, A. Georges, and I. Verbauwhede, "Exploiting hardware performance counters," in *Proc. 5th Workshop Fault Diagnosis Tolerance Cryptogr.*, Aug. 2008, pp. 59–67.

[151] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing performance monitors for compromising keys of RSA on Intel platforms," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* New York, NY, USA: Springer, 2015, pp. 248–266.

[152] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*. New York, NY, USA: Springer, 2015, pp. 48–65.

[153] M. Payer, *HexPADS: A Platform to Detect 'Stealth' Attacks*. Cham, Switzerland: Springer, 2016, pp. 138–154.

[154] J. Nomani and J. Szefer, "Predicting program phases and defending against side-channel attacks using hardware performance counters," in *Proc. 4th Workshop Hardw. Archit. Support Secur. Privacy (HASP)*, 2015, p. 9.

[155] B. Gulmezoglu, A. Zankl, T. Eisenbarth, and B. Sunar, "PerfWeb: How to violate Web privacy with hardware performance events," in *Proc. Eur. Symp. Res. Comput. Secur.* New York, NY, USA: Springer, 2017, pp. 80–97.

[156] G. Irazoqui, "Cross-core microarchitectural side channel attacks and countermeasures," Ph.D. dissertation, Dept. Electron. Commun. Eng., Worcester Polytech. Inst., Worcester, MA, USA, 2017.

[157] V. Weaver and J. Dongarra, "Can hardware performance counters produce expected, deterministic results," in *Proc. 3rd Workshop Functionality Hardw. Perform. Monitor.*, 2010, pp. 1–27.

[158] J. Dongarra, K. London, S. Moore, P. Mucci, D. Terpstra, H. You, and M. Zhou, "Experiences and lessons learned with a portable interface to hardware performance counters," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, p. 6.

[159] T. Mytkowicz, P. F. Sweeney, M. Hauswirth, and A. Diwan, "Time interpolation: So many metrics, so few registers," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2007, pp. 286–300.

[160] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-Inspired Inf. Commun. Technol.*, 2016, pp. 21–26.

[161] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2015, pp. 11–20.

[162] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in Android malware detection," in *Proc. ACM Conf. SIGCOMM*, vol. 44, 2014, pp. 371–372.

[163] T. Alpcan and T. Basar, "A game theoretic approach to decision and analysis in network intrusion detection," in *Proc. 42nd IEEE Int. Conf. Decis. Control*, vol. 3, Dec. 2003, pp. 2595–2600.

[164] M. Kodialam and T. V. Lakshman, "Detecting network intrusions via sampling: A game theoretic approach," in *Proc. 32nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, 2003, pp. 1880–1889.

[165] J. Gomez and D. Dasgupta, "Evolving fuzzy classifiers for intrusion detection," in *Proc. IEEE Workshop Inf. Assurance*, New York, NY, USA, vol. 6, Jun. 2002, pp. 321–323.

[166] S. M. Bridges and R. B. Vaughn, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in *Proc. 12th Annu. Can. Inf. Technol. Secur. Symp.*, 2000, pp. 109–122.

[167] H.-D. Huang, G. Acampora, V. Loia, C.-S. Lee, and H.-Y. Kao, "Applying FML and fuzzy ontologies to malware behavioural analysis," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jun. 2011, pp. 2018–2025.

[168] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. New York, NY, USA: Springer, 2015, pp. 3–25.

[169] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3332–3344, Nov. 2016.

[170] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *Proc. 54th Annu. Design Autom. Conf. (DAC)*, 2017, p. 25.

[171] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu Ghazaleh, and D. V. Ponomarev, "EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers," *IEEE Trans. Dependable Secure Comput.*, early access, Feb. 5, 2018, doi: 10.1109/TDSC.2018.2801858.

[172] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[173] R. Gibbons, *A Primer Game Theory*. Harvester, U.K.: Wheatsheaf, 1992.

[174] J. Yen and R. Langari, *Fuzzy Logic: Intelligence, Control, and Information*, vol. 1. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[175] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices.," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 980, Jun. 2016.

[176] R. Guanciale, H. Nemati, C. Baumann, and M. Dam, "Cache storage channels: Alias-driven attacks and verified countermeasures," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 38–55.

[177] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "ASLR on the line: Practical cache attacks on the MMU," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, p. 13.

[178] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in JavaScript and their implications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 1406–1418.

[179] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke, "CacheAudit: A tool for the static analysis of cache side channels," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, pp. 1–32, Jun. 2015.

[180] S. Wang, P. Wang, X. Liu, D. Zhang, and D. Wu, "CacheD: Identifying cache-based timing channels in production software," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 235–252.

[181] Y. Xiao, M. Li, S. Chen, and Y. Zhang, "STACCO: Differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 859–874.

AYAZ AKRAM received the bachelor's degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and the master's degree in computer engineering from Western Michigan University, USA. He is currently pursuing the Ph.D. degree in computer science with the University of California, Davis, Davis, CA, USA. His research interests include computer architecture, high-performance computing, and the intersection of computer architecture with other areas like computer security and machine learning.

MARIA MUSHTAQ received the Ph.D. degree from Lab-STICC, University of South Brittany (UBS), France, in 2019. She is currently a Postdoctoral Researcher with the LIRMM, University of Montpellier (UM), France. She has specific expertise in developing runtime detection and mitigation solutions against side-channel information leakage in computing systems. Her research interests mainly focus on cryptanalysis, constructing and validating software security components, and constructing OS-based security primitives against various hardware vulnerabilities.

MUHAMMAD KHURRAM BHATTI received the M.S. and Ph.D. degrees in embedded systems from the University of Nice-Sophia Antipolis, France. He was a Postdoctoral Researcher with the KTH Royal Institute of Technology, Stockholm, Sweden. He is currently an Assistant Professor with Information Technology University, Lahore, Pakistan. His research interests include embedded systems, information security at both hardware and software level, cryptanalysis, mixed criticality systems, and parallel computing systems.

VIANNEY LAPOTRE received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University Bretagne Sud, France. He spent six months as an invited Researcher at the Ruhr-University of Bochum, Germany. He was a Postdoctoral Researcher with LIRMM, Montpellier, France. He is currently an Associate Professor with University Bretagne Sud. His research interests include hardware security, and reconfigurable and self-adaptive multiprocessor architectures.

GUY GOGNIAT is currently a Professor of ECE with the University of Bretagne-Sud, Lorient, France, where he has been since 1998. In 2005, he spent one year as an Invited Researcher at the University of Massachusetts Amherst, Amherst, MA, USA, where he worked on embedded system security using re-configurable technologies. His work focuses on embedded systems design methodologies and tools. He also conducts research in the domain of reconfigurable and adaptive computing and embedded system security.