IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 April 2023, accepted 4 May 2023, date of publication 12 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275757

SURVEY

Microarchitectural Side-Channel Threats, Weaknesses and Mitigations: A Systematic Mapping Study

ARSALAN JAVEED, CEMAL YILMAZ, (Member, IEEE), AND ERKAY SAVAS, (Member, IEEE)
Faculty of Engineering and Natural Sciences, Sabanci University, 34956 Istanbul, Turkey
Corresponding author: Arsalan Javeed (ajaveed@sabanciuniv.edu)

ABSTRACT Over the course of recent years, microarchitectural side-channel attacks emerged as one of the most novel and thought-provoking attacks to exfiltrate information from computing hardware. These attacks leverage the unintended artefacts produced as side-effects to certain architectural design choices and proved difficult to be effectively mitigated without incurring significant performance penalties. In this work, we undertake a systematic mapping study of the academic literature related to the aforementioned attacks. We, in particular, pose four research questions and study 104 primary works to answer those questions. We inquire about the origins of artefacts leading up to exploitable settings of microarchitectural side-channel attacks; the effectiveness of the proposed countermeasures; and the lessons to be learned that would help build secure systems for the future. Furthermore, we propose a classification scheme that would also serve in the future for systematic mapping efforts in this scope.

INDEX TERMS Cybersecurity, microarchitecture, side-channel, systematic-mapping.

I. INTRODUCTION

The manifestation of computer architecture roots into microarchitecture and instruction set architecture (ISA). Microarchitecture refers to the ways, in which an ISA is implemented for a specific microprocessor. For reasons, such as design, cost, and optimization, a given ISA can be implemented under different microarchitectures, which typically distinguish among each other in the way constituent components of the processors are interconnected and interoperated. With all these aforementioned complex entanglements in play, exploitable security vulnerabilities may arise, which can be abused by a malicious adversary to affect the confidentiality and integrity of a computing system, potentially resulting in serious loss.

In recent years, among some of the many reported side-channel attacks exploiting the microarchitectural vulnerabilities, Meltdown [1] and Spectre [2] are few of the notable and widely talked-about examples, as these attacks particularly abuse the transient state of the microprocessors during speculative execution. The mechanics of these attacks are although complex, yet they are practical and have successfully demonstrated exfiltration of sensitive private data and cryptographic secrets.

Furthermore, the findings about the microarchitecture attacks have sparked the interest of the security research community and accelerated the research efforts in this relatively recent area of software and cyber security. In this regard, research efforts are being carried out both on the axis of uncovering newer vulnerabilities and to prevent, mitigate, and lessons to build secure systems of the future.

The literature on microarchitecture security research has, therefore, been growing at a relatively fast pace. Typically a microarchitecture side-channel attack is discovered and reported at first, followed by research efforts into its potential attack variations, mitigations, and security-oriented lessons to be learned. However, efforts that over-arch the individual works to systematically gather and to assemble various aspects of microarchitecture security research in a broader context, are yet limited and somewhat outdated, and thus

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

are still desired. The purpose of our presented work as a systematic mapping study is a step in this direction. To the best of our knowledge, this work is the first systematic mapping study carried out on this subject.

A systematic mapping study provides an overview of the existing research through systematic classification of the published literature on the topic, guided by the posed research questions under the constraints of a mapping protocol [3]. A prime finding from such studies are the research spots, which have gained considerable attention, as well as the emerging research trends, which can launch newer investigations and guide the direction for further research. In contrast, a literature survey aims to comprehensively index and, thus report the existing state of affairs on a topic of interest. Consequently, systematic mapping studies and surveys differ from each other in terms of their end goals and the research process they follow.

To guide our investigation, we have raised four research questions (RQs)(Section IV-A) to address the security concerns about microarchitectures, such as the aspects leading up to leakage, the ways in which exploitable corridors are created, the effectiveness of the existing countermeasures as well as their applicability to zero-day attacks, and the lessons to build more secure systems.

The main contributions of our mapping study are: i) a broad overview of the research efforts in the area of microarchitecture security; ii) a classification scheme that can be used to classify newer research in this area; iii) a decent yet diverse coverage of the published research on the topic, particularly from the last five years; and iv) the identification of hot and cold research spots to date.

The target audience of this work is the researchers and practitioners, who aim to establish a better overarching understanding of the research efforts being carried out in this area.

The remainder of the paper is organized as follows: Section II discusses related work; Section III presents the necessary background information; Section IV presents the methodology followed, detailing the search, inclusion, and the exclusion criteria together with the study selection process used; Section V introduces the classification scheme employed, followed by the mapping results; Section VI details the answers to our research questions; Section VII presents potential threats to validity in the study; and Section VIII concludes the paper and presents possible directions for future work.

II. RELATED WORK

To the best of our knowledge, our study is the first systematic mapping study in the area of microarchitectural side-channel threats, weaknesses, and mitigations. However, there exist a few surveys in the literature [4], [5], [6] as well as some individual works [7], [8], [9], each addressing a specific aspect. Albeit, some of these works are either being somewhat dated or generally lacked wider scope. Table 1 presents a highlevel glimpse on existing related work together with its strengths and limitations.

TABLE 1. Publication year, Strengths(S) and Limitations(L) of notable related work.

Article	Year	Standing	
[4]	2012	S	covers a range of attacks circa. 2012
		L	somewhat outdated and exclusive focus on embedded micro-processors
[5]	2018	S	variety of microarchitecture attacks and defenses presented with emphasis on role of cache memory
		L	devoid of attacks involving prefetchers
[6]	2018	S	focus on timing attacks
		L	lacking attacks employing other vectors

Kanuparthi et al. present a survey [4] dating till 2012. The scope of the survey is bound to embedded microprocessor security only. As the proliferation of embedded systems was on the rise, so as emerging challenges to secure their vulnerable nature due to their limited onboard resources. The authors reported some of the prominent attacks plaguing that arena during those times. Furthermore, the authors compared the trade-offs among sought-after countermeasure approaches, such as levering integrity checks, data encryption, and microarchitecture revisions.

Szefer et al. present a survey [5] on microarchitectural side- and covert-channels with their defense proposals dated until 2018. The prime emphasis of the aforementioned work is on timing-based and (to some degree) access-based exploitations. In a distinct regard, the authors did foresee the role of the prefetchers in microarchitectural exploitations and did theoretically present a setting for such an attack, yet were unable to provide a citation to any known published literature, since the research into prefetcher exploitations was still in its infancy during the time of the work.

Another survey [6], which Ge et al. authored around the same year of 2018, exclusively focused on the timing-based microarchitectural attacks and their countermeasures. The main motivation behind this survey was to taxonomize the aforementioned efforts. These attacks established their reputation as prime vectors for remote exploitation in cloud computing. Furthermore, the authors emphasize the need to secure cloud systems against such class of attacks, as our ever-increasing adoption and growing reliance on cloud computing would continue to grow with the passage of time.

Note that a direct comparison with these works wouldn't be straightforward because of the different sought after research objectives as well as the methodology being employed. Last but not least, we believe the constituent studies employed in our work are fairly recent in the sense that half of them have been published in the last five years (since 2018).

As we and other researchers [5], [6] have observed that microarchitecture security is a fast-growing field and even the entire playground may change quickly. So we believe, in this time frame, our systematic gathering of literature, especially from the last five years is going to be an important effort and would serve as a valuable scientific resource.

III. BACKGROUND INFORMATION

To assist the understanding of the reader for the remainder of this article, we, in this section, introduce essential terms/concepts and provide some relevant definitions for the material ahead. Furthermore, for clarity, we also provide Table 6 in Appendix B to list out all the abbreviations and their expansions in one place, which will be used throughout this article.

The usual intention behind a microarchitectural side-channel attack undertaken by a *malicious actor* is to exfiltrate some *secret* information or to perform acts of sabotage on a system. In literature, malicious actors are also often referred to as *adversaries*. This secret information is usually well-protected and is not meant to be revealed to any adversaries. In the case of cryptography, this secret information could be a secret key, the content of a file containing sensitive business information, the content of a buffer, or the latent parameters of a machine learning algorithm. In contrast, an act of sabotage would intend to disrupt the services provided by a system, such as a ransomware attack. Regardless of the intention of the malicious actor, an affected system or a process is generally referred to as a *victim*. If the malicious actor's main intention is to exfiltrate some information then it is referred to as a *spy* in contrast to a *saboteur* carrying out a sabotage. Both the spy and saboteur could be a single process, a group of processes, a compromised system component, or another involved remote actor.

A *side-channel* is a communication channel established between a sender party and one or more receiver parties, such that sensitive information is unintentionally leaked as a result of side-effects to computations being carried out on the underlying hardware. In contrast, a *covert channel* is a side-channel that is intentionally established between a malicious sender and receiver, upon which exfiltrated secret information is being transmitted.

A CPU (central processing unit) also referred to as *processor* or *microprocessor*, executes the machine instructions. A CPU is realized on a silicon die through an electronic design and automation (EDA) process and usually comprises billions of interconnected transistors. A CPU provides a programmable interface in the form of *instruction set architecture* (ISA). The ISA could be implemented in several ways dictated by the underlying microarchitectural design, i.e. *microarchitecture*. Generally, a microarchitecture comprises some specialized processing elements, which are referred to as *functional units*, and some intermediate temporary storage elements, which are referred to as *registers* and *buffers*. A register is a small, but fast temporary storage area within the processor to hold some data, such as an argument to an instruction. A buffer, on the other hand, is a temporary storage area allocated in the main memory to hold data in-between transfers. In recent microarchitectures, however, few specialized buffers exist in the CPU to assist the operations of various onboard functional units, such as pipeline buffers. Furthermore, a CPU or GPU (graphics processing unit) could be equipped with *performance counters*, which are special onboard registers counting the low levels occurring in the CPU, such as the number of cache misses or hits. A GPU is a specialized processor meant to process computer graphics.

Instructions of a program execute in a pipeline and typically go through instruction- *fetch*, *decode*, *execute*, *memory*, and *writeback* steps. During each stage of this pipelined execution, the intermediate results are stored in the pipeline buffers where the arguments and the operational configuration are provided through intermediate registers.

The microarchitecture itself is a specific instance of the number of valid ways, in which the underlying *functional units* of the CPU should be interconnected to fulfill the design and performance goals intended by the designers. On modern CPUs, the ISA is mapped onto a sequence of small microcode operations established by the underlying microarchitecture. Instructions are executed in one or more clock ticks, which are referred to as *clock cycles*.

A multicore processor is a single processor containing multiple execution cores in a single package. Each core has some core-private resources, such as cache and pipeline stages, in addition to having some resources shared with other cores. Multicore processors deliver significant performance improvements as the workload can be distributed among available cores to be executed concurrently.

Out-of-order and speculative execution are other performance enhancement mechanisms, leveraged to optimize the execution of instruction stream on modern processors. *Out-of-order execution* refers to executing instructions independently from each other in their order of execution to increase overall throughput. Those instructions that do not have pending data dependencies are executed immediately whereas instructions with pending dependencies are scheduled for a later time when those dependencies are met. However, out-of-order execution remains hidden from the application point of view and everything appears as it has been executed sequentially. In a distinct regard, *speculative execution* refers to executing branches speculatively ahead of time by guessing the most probable branch. If the guesswork is correct then results are already available from the executed branch. Otherwise, the executed branch is discarded and the correct branch is taken.

Caches are small memories that bridge the data-access latency between slow but large physical memory and fast CPU(s) by buffering frequently accessed data. Modern systems typically have multiple cache levels (L1, L2, and often L3) arranged in a hierarchy comprising the *cache-subsystem*. L1 cache is split as instruction and data cache, followed by a larger L2 cache. L1 and L2 caches are core-private, i.e., each core gets its own dedicated set of these caches. On some systems, L3 cache, also called the last-level cache (LLC), is shared between all the CPU cores. On modern Intel microarchitectures, LLC is inclusive in nature [10], which contains all the data within L1 and L2 caches. Given a memory address to be accessed, the CPU first looks it up in the L1 cache, if the respective data is found in the cache, its called a *cache hit*, otherwise a *cache miss*. Upon a cache miss, the CPU next looks for the address in next level

cache and keeps on traversing the cache hierarchy until either the respective data is located in the cache or it reaches all the way to the main memory. A cache is organized into *cache lines* and *cache sets*, each of which contains the same number of cache lines. A given memory address is decoded into *tag*, *index*, and *offset* fields in order to look up the respective data in the cache hierarchy.

The memory controller handles the memory accesses to DRAM when data requested by the CPU is not available in the cache. *Memory controller* serves a data request by translating the given physical address into an internally maintained DRAM map of channel, bank, row, and column information. Furthermore, the memory controller arbitrates internally among the concurrent memory accesses through scheduling and buffering policies. *Memory deduplication* is a mechanism to save the physical memory space by keeping one copy for each duplicated memory pages as long as the pages are unmodified. The modified page is isolated and kept as a separate independent page.

Prefetching is among the few employed methods besides speculative execution to enhance performance in modern CPUs. Hardware prefetchers are those microarchitectural units that perform the task of prefetching and handling associated aspects. In a nutshell, prefetchers increase the cache hit rate through speculatively prefetching the data that would potentially be requested in the near future during a computation. To this end, patterns of memory access requests are observed and speculative data fetching is issued accordingly. Prefetchers exist in a hierarchy among all the levels of cache-to-memory, such as L3-L2 and L2-L1. The implementation details of the prefetchers are usually proprietary and subject to change as the underlying microarchitecture evolves. Broadly speaking, prefetchers are categorized into *next line prefetchers* and *stride prefetchers*. The former simply prefetches the next line for the current cache-line in use, whereas the latter learns and utilizes the patterns in memory accesses.

In recent years, a paradigm in secure computing has emerged themed around its reliance upon hardware-based *Trusted Execution Environments* (TEE), realized in software through enclave programming [11]. CPU vendors provided TEE environments through microarchitectural extensions. For instance, ARM provides ARM-TrustZone [12] and Intel provides TPM+TXT [13] and later Software Guard Extensions (SGX) [14].

Last but not least, *Peripheral Component Interconnect* (PCI) and its modern high-speed variant PCIe (PCI *express*) is the de-facto protocol with supporting hardware, through which CPU and high-speed peripheral devices, such as GPU and NIC, are connected.

IV. METHODOLOGY

In this work, we followed the widely-accepted guidelines for carrying out systematic mapping studies [15], [16], which had also been adopted by some of the relevant mapping studies in the literature [17], [18], [19]. To this end, we first carried out a planning phase, primarily for laying out the research questions, choosing the method to locate and appraise primary studies, crafting a search strategy, and carrying out the curation of the primary studies. In this context, a primary study is a chosen, peer-reviewed, and published scientific article that went through the usual phases of searching, screening, and classification, which are performed as a part of the typical workflow in a systematic mapping process. Each primary study is meant to contribute to the systematic data synthesis by answering one or more research questions.

We, in particular, decided to follow a mapping process comprised of the following steps [15]: *defining research questions*, *conducting search*, *screening of papers*, *keywording of abstracts*, *classification*, and *data extraction and mapping*. Although all of these steps are sequential in nature and performed one after another, oftentimes the individual steps are performed iteratively more than once to refine their outcomes and improve the overall end result of the whole mapping process. Next, we discuss how we actually carried out each of these steps in this study.

A. DEFINING RESEARCH QUESTIONS

We put together the following research questions as a basis for this study:

- RQ1: What are the aspects of microarchitectural artefacts, which contribute to the sensitive information leakage to compromise security and privacy?
- RQ2: How some of the recent microarchitectural surfaces were crafted and turned into feasible attack corridors?
- RQ3: How effective are the proposed countermeasures of microarchitecture side-channel attacks and whether these countermeasures are generalizable? Can these generalized countermeasures predict/prevent zero-day attacks?
- RQ4: Given the published countermeasures, how secure a system we can build against microarchitecture side-channel attacks and what lessons we can incorporate in this system-design process?

B. CONDUCTING THE SEARCH

Guided by our research questions, we defined an initial set of keywords as seeds to locate relevant papers in Google Scholar [20]. We read these papers and utilized *snowballing technique* [21] to find additional papers that we find relevant to our RQs. Snowballing refers to locating additional papers based on the reference lists or citations of a given paper [18]. To aid the snowballing process, we used an online tool, called Connected Papers [22]. In particular, we came up with 11 core papers as a result of the initial search, these papers were used to put together a set of keywords (listed in Table 2), which were then used as the basis for defining search strings. The validity of these keywords was tested by rediscovering the core papers they were driven from. We also eliminated some keywords that we found to be redundant, superfluous,

TABLE 2. Keywords used to form search strings. To avoid redundancy, we are adopting regex format to specify potential variations of a keyword. We used these keywords in *Title, Abstract, Author defined,* and *All metadata* fields for performing database queries.

Keyword(s)
"micro[-]?architecture", "attack", "vulnerability", "side[-]?channel", "origin[s]?", "defen[c|s]e", "counter[-]?measure[s]?", [0|zero]?[-]?day, "detect[ion]?","prevent[ion]?", "leakage", "predict", "secure", "system"

or causing an increased number of irrelevant results in the search.

To carry out the actual search, we utilized three standard databases as sources, namely *IEEExplore* [23], *ACM Digital Library* [24], and *Springer Link* [25]. This decision was primarily influenced by the availability of the database subscriptions that our campus had, the manpower of our team, and the fact that these sources are the de-facto prime venues of scientific literature in computer science. Furthermore, we leveraged the advanced search features of these databases to perform the search queries. The advanced search features enabled us to limit or expand the scope of the queries for decreasing the frequency of the irrelevant results. Based on the keywords given in Table 2, the search strings for each database were created and iteratively experimented with to come up with the most appropriate search results.

We, furthermore, observed that the internal mechanics of these databases were different. That is, a query suitable for one database was not necessarily to be equally effective in another database. For example, we found out that searching for keywords in the abstracts at IEEExplore was more effective than running the same search at ACM Digital Library. Nonetheless, we used the listed set of keywords (Table 2), the database-specific advance search features (e.g., boolean operators), and available filters to tailor the relevance of the search results.

After getting the search results, we sorted them in the order of relevance and picked the top 100 results for ACM Digital Library and Springer Link. For IEEExplore, we chose to stop when 5 irrelevant papers in a row were observed. We used ranking by relevance for ACM and Springer Link databases. For IEEExplore, however, this feature was unavailable. Thus, we opted to use two different aforementioned stopping criteria. Other works [18], [26] have also followed a similar approach.

C. SCREENING AND CLASSIFICATION PROCESS

To aid the screening and the study selection process, we tailored the following inclusion and exclusion criteria, which were adopted from [16], [19], and [18].

1) INCLUSION CRITERIA

- a study must answer at least one of the formulated research questions (RQs),
- a study must be dated until the March of 2023,
- priority must be given to the evaluation and validation research studies during the selection process,
- a study must be peer-reviewed and published in English.

2) EXCLUSION CRITERIA

- a study should not be in one of these forms: patents, white papers, reports, thesis, tutorials, and webpages,
- duplicate studies located by different search engines,
- inaccessible, irretrievable, or irrelevant to the theme of our work,
- any article, the subject matter of which is around quantum computing.

Note that the aforementioned criteria aided us to perform the initial screening and sanitation of the search results.

D. OVERVIEW OF THE DATASET

Our initial search effort resulted in 546 articles from the three databases as follows: 192 from IEEExplore, 210 from ACM Digital Library, and 144 from Springer Link. However, after the title and abstract screening, we manually went over the content of the articles, i.e. *content scanning*, especially for those articles where the abstracts were short enough to be suitably meaningful or lacked ample clarity in general. Furthermore, we also content scanned those articles with abstracts where it was found difficult to pinpoint the investigated problem and/or the main contribution being made. We also performed content scanning of the individual sections for the papers, where we felt that individual clauses of inclusion and exclusion criteria were difficult to apply. In such a case, we relied on team discussion to reach a final consensus. Other mapping studies [18], [27] have also followed a similar practices.

For classification we settled for 104 articles to serve as our primary studies, the list of which is available in Table 5, whereas, the remainder 442 (out of 546) articles were dropped at this step. The classification scheme is described in Section V which guided us toward the final data extraction and mapping of results. Ultimately, the primary studies also served as the basis for answering our RQs, which was, indeed, the main goal of this mapping study.

Regarding further relevant statistics of the primary studies dataset, we notice that 25% (25 out of 104) of articles are published in journals and remainder 75% (79 out of 104) as conference papers. Similarly, page count of primary studies span between 4-27 pages, where 12 is the median count. Lastly, 57% (60 out of 104) of primary studies primarily focused on a countermeasure approach and remainder 43% (44 out of 104) proposed an attack approach. These aforementioned statistics establish a fair degree of confidence about the balance in the dataset for our systematic mapping study.

V. RESULTS

Systematic mapping studies aim to provide an overview of a research arena through the classification of published literature on the subject topic. To aid discussion and present the results of our study, we first describe the classification

scheme, which we used to categorize the primary studies. We then present the mapping results in the light of the aforementioned scheme.

In accordance with the established norm of systematic mapping studies, we first describe the classification scheme, followed by the presentation of aggregate statistics on the individual facets of classification. The main takeaway from the aggregate statistics is the quantitative presentation of the data gathered from primary studies. This data presents an overall picture and helps build ample confidence in the selection and the classification process as well as in the thematic diversity present in the pool of primary studies. The primary studies are then used to answer the main research questions in this study.

A. CLASSIFICATION SCHEME

Our classification scheme broadly classifies the studies into seven categories: *research type*, *main contribution*, *hardware platform*, *instruction set architecture (ISA)*, *leakage vector*, and *leakage component*. Next, we describe each of these categories.

1) RESEARCH TYPE

This category describes the different research approaches as outlined by Wieringa et al. [28]. We chose to use Wieringa's classification scheme as it has also been used in earlier mapping studies [18], [19], [29]. We associated each of our primary studies with one of these research approaches: *evaluation*, *validation*, and *solution proposal*. Based on a typical exclusion criterion of mapping studies [18], [26], we disregarded *philosophical*, *opinion*, and *personal experience* papers, and focused on opting primarily for the *evaluation* and *validation* research. Note that the aforementioned types of research aim to demonstrate the practical usage of a technique backed by real data, relevant experiments, and empirical evaluation.

2) MAIN CONTRIBUTION

Categorization based on contribution type in mapping studies has been employed previously by Zein et al. [19] and Shahrokhni et al. [17], which classifies each primary study to one of the following specific types of contribution: *framework*, *tool*, *metric*, *approach*, and *criterion*.

A *framework* paper introduces a detailed method, which has a broad scope and tends to focus on more than one research question or area. Whereas, an *approach* paper has a narrower scope and tends to have a more specific goal addressing a single research question. Similarly, a *tool* paper presents an implementation of at least one approach with the aim of demonstrating the applicability of the proposed approach to the practitioners. Primary studies, where the core emphasis was on tool demonstration, are classified in this category. A *metric* paper, on the other hand, proposes an empirical measure to quantitatively describe a variable of interest. In contrast, a *criterion* paper outlines a strict method, by which a certain quantitative or qualitative attribute of an observatory aspect can be judged upon.

3) HARDWARE PLATFORM

This category represents the nature of the physical hardware considered by a primary study, upon which the main subject matter was presented and/or evaluated. This categorization would help establish whether microarchitecture attacks are specific to or prevalent for a certain hardware type. We opted for the following subcategories: *commodity*, *mobile*, *embedded/IoT*, *cloud*, and *any*, based upon the explicit mentioning of the platform type discussed. *Commodity* hardware refers to the general purpose computing platforms, such as workstations for daily usage. Whereas, *mobile* hardware refers to the handheld portable devices, which are primarily meant for communication and serve the limited computing needs of the users. Smartphones are a prime example of this category. Similarly, *embedded/IoT* devices refer to the microprocessor-equipped hardware with optional networking ability and occasional connections to the internet, which are designed to perform a dedicated function. Similarly, *cloud* platform refers to a group of server machines that are networked together and connected to the internet, located typically in a data-center environment. The users are given remote access over the internet to utilize these machines for their application usecases. Last but not least, the *any* category refers to either an unspecified platform or the specification of the platform does not matter. Those primary studies, which present a generic approach, criteria, or empirical metric, fall under this category as their subject matter is agnostic to a specific hardware type.

4) EVALUATION METHODOLOGY

This category refers to the methods adopted in the primary studies to conduct their experimental evaluations. In this regard, we have observed the following types of methodologies: pure *software* implementation, employment of *simulation* tools, evaluations performed on *real* hardware, evaluations performed on *FPGA*-synthesized hardware, and *other*, referring to the studies where the aforementioned categories do not matter. For instance, studies presenting a criterion or an empirical metric belong to the *other* category. Note that primary studies may follow more than one evaluation methodology, such as a study carrying out the evaluations through simulation followed by hardware implementation and so on.

5) INSTRUCTION SET ARCHITECTURE

Instruction set architecture (ISA) refers to the syntactic and semantic realization employed at the machine level, to which the compiled software is assembled for execution. In this regard, we observed that the following types of ISAs were used in the primary studies selected: *Intel*(x86,x86_64), *ARM*, *RISC*, Nvidia-*CUDA*, *MIPS*, and *other*, referring to the studies where either multiple ISAs or lesser-known ISAs are used. This categorization would help establish whether microarchitecture attacks are agnostic to ISA or not.

6) LEAKAGE VECTOR

Leakage Vector refers to the side-channel, through which the sensitive information leaks or emanates. In this regard,

we consider the following leakage vectors: *access patterns*, *contention*, *sharing*, *dependence*, *duplication*, *EM*, *execution*, *fault*, *interruption*, *power*, *speculation*, *state*, and *timing*.

Access patterns refer to the recurrent patterns observed in accessing some hardware resources, through which the high-level behavior of the requesting entities can be inferred. This can later be used as a ladder step to either assist or rely for further malicious exploitation. In contrast, *contention* refers to at least two entities contending to get access to a shared resource to perform their task. Similarly, *sharing* refers to a shared resource being granted (according to some agreed-upon policy) to one of the several requesting entities. Whereas, *dependence* refers to an entity being reliant upon some resources or being controlled by some resources, to perform its assigned function. In contrast, *duplication* refers to creating a duplicated copy of a requested resource for the requesting entity to fulfill its needs. Moreover, *EM* refers to the electromagnetic emanations, which can be observed at a distance without physical proximity. These emanations can be utilized to infer the underlying state as well as the ongoing activities of the system radiating from them. Similarly, *power* refers to the electrical power being consumed in relation to the hardware demand. Whereas, *execution* refers to the act of executing instructions on a processor. *Fault* refers to an event or untimely interruption, which disturbs the sanity of the internal state of a processor and the correctness of an ongoing computation. To recover from a faulty state, often the intermediate computation is discarded and the execution is rolled back to some previously known good state. In contrast, *interruption* refers to an event, which needs to be served immediately, causing temporary suspension of the lower-priority execution happening at the time of the interruption. *Speculation* refers speculative execution (Section III). Similarly, *state* refers to an existing internal state of a resource, which would lead to one of the possible future states. Lastly, *timing* refers to the elapsing of the time between two microarchitectural events of interest.

7) LEAKING COMPONENT

This refers to the microarchitectural functional unit or subsystem, which is the culprit of the side-channel leakage by exposing one or more leak vectors. The constituents for this category that we relied upon organizing the selected primary studies are: *buffers*, *caches*, *instructions*, *interconnects*, *memory*, *microprocessing elements*, *tee* (trusted execution environment), and *timers*. The background information for these components has been presented in Section III

B. MAPPING RESULTS

In this section, we present the quantitative and qualitative results of our systematic mapping study with respect to the classification scheme introduced in Section V-A.

1) PUBLICATIONS TIMELINE

We curated a total of 104 primary studies by following the steps of the systematic mapping methodology introduced in Section IV. Table 5 lists the title of each study and its associated bibliographic reference. Furthermore, each study is assigned a sequential study ID, such as S1, S2, and S3. We use these study IDs to refer to specific primary studies throughout the remainder of the paper. Note that assigning study ID is an established practice and has widely been followed in other mapping studies [19], [26]. More specifically, study IDs help the reader differentiate between references to bibliographic items versus a primary study.

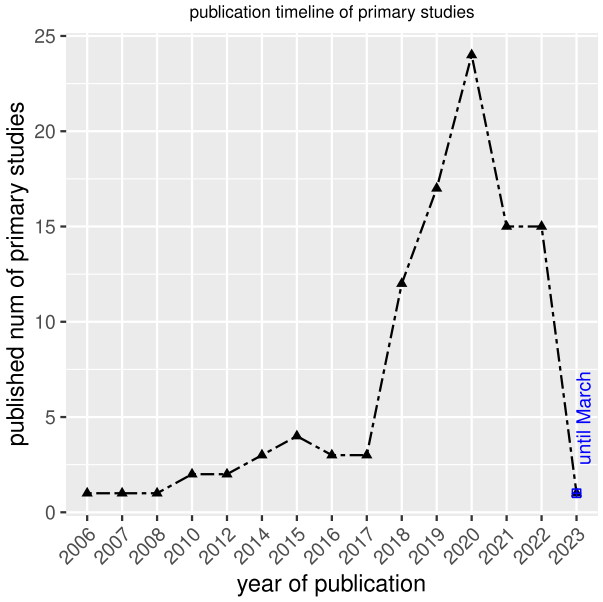

FIGURE 1. Publication timeline of the primary studies. The timeline spans from 2006 to 2023. For each year, the number of primary studies is indicated on the vertical axis. The year 2023 is covered only until month of March.

Figure 1 illustrates the publication timelines for the primary studies over the years from 2006 to 2023. We first observe that the research in microarchitectural side-channel attacks became visible in 2006 and gained gradual attention over the years till 2017. From 2017 onward, the research area gained more and increasing attention, which is evident from the sharp increase in the number of publications. Software and cyber security has been gaining more and more attention over the recent years and this trend has been observed in other studies [18], [26] as well.

Figure 1 also illustrates that the pool of primary studies we assembled for this work is fairly recent. More specifically, 83% (87 out of 104) of the studies were published in the last 6 years (i.e., between 2017 and 2022). Note that, for the year 2023, the studies were included until the end of March, during which we carried out the analysis.

2) RESEARCH SPOTS

Figure 2 highlights the research spots as a matrix of bubbles where the size of a bubble is proportional to the volume of the studies at the intersection of the respective leakage vector and leaking component categories. As such, the figure reveals the hot and cold spots where the research has focused.

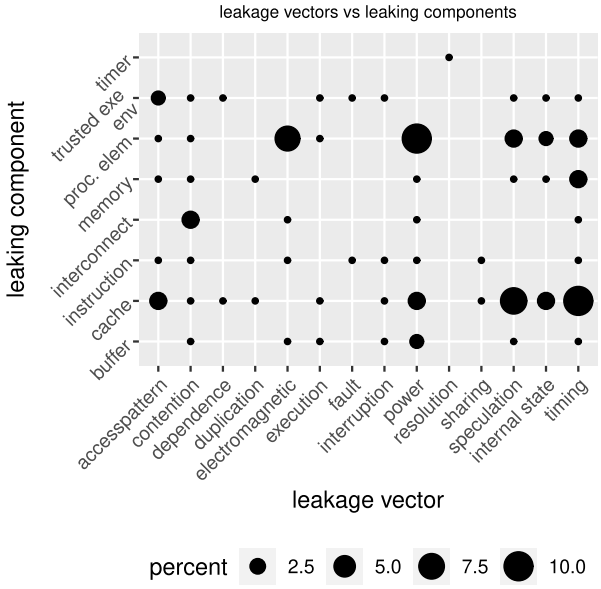

FIGURE 2. Research spots for leakage vector vs. leaking components.

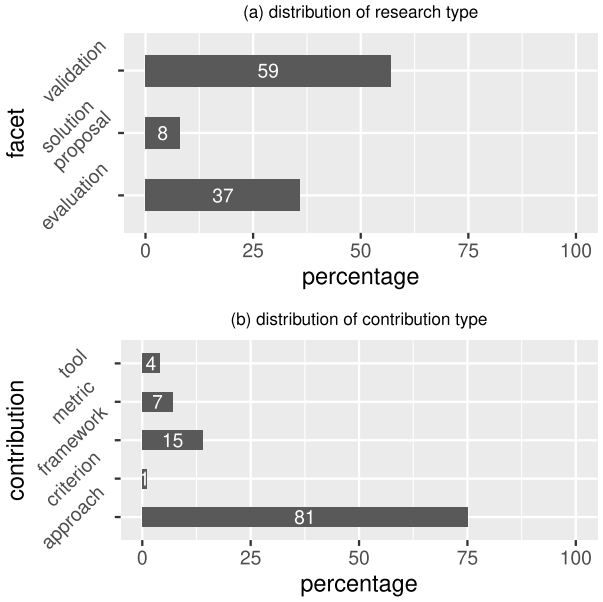

FIGURE 3. Percent-wise share of the primary studies with respect to (a) research facet and (b) main contribution. The length of a bar depicts the percentage value, whereas the integer associated with the bar indicates the actual count.

Although some care must be taken when interpreting the research spots in Figure 2 as the pool of primary studies used in the analysis is solely assembled based on a strict selection criterion designed to address our research questions (Section IV-C), the figure illustrates some important research characteristics of the primary studies. Among the notable patterns, we observe that *contention* and *timing* as the leakage vectors have been involved across almost all (7 out of 8) the leaking components. Similarly, *cache* and *trusted execution environment* as the leaking components have been involved across most (11 and 9 out of 14, respectively) of the leakage vectors. The ways, in which the various leaking components leak information through leakage vectors are discussed at length in Section VI.

3) RESEARCH FACET AND CONTRIBUTION TYPE

We classify the primary studies (Table 5) along the axes of research facet and main contribution (Section V-A). Figure 3 illustrates the results we obtained. We observe that most of the primary studies belong to the *validation* category (59 out of 104), followed by the *evaluation* category (37 out of 104), and then the *solution proposal* category (8 out of 104). Note that the *solution proposal* category is customarily [18] excluded from the study selection criteria, because, by nature, they lack considerable evaluation. However, we opted to flex on this practice and, in the final iteration of our study selection process, we included a small number (i.e., 8) of solution proposals. We did this because we found the presented ideas to be interesting and relevant to the scope of our research questions.

Along the same lines, we observe from Figure 3(b) that 75% of the primary studies are actually contributing approaches backed by evaluations. However, the remaining 25% contribute *tools* (4 out of 104), *metrics* (7 out of 104), *frameworks* (15 out of 104), and criteria (1 out of 104). Note that the studies occasionally contributed to more than one category.

4) TARGET PLATFORM AND INSTRUCTION SET ARCHITECTURE

In terms of the target platform hardware and the instruction set architecture (Figures 4a and b, respectively), we observe that all of our primary studies have considered them all, albeit to various degrees as shown through percent-wise proportion in Figure 4. However, we do observe that the commodity hardware platform and the Intel x86 ISA have the largest share overall, which is to be expected due to their proliferation worldwide. Nonetheless, we see contributions from other ISAs and platforms too, which reveals an important fact that microarchitectural exploitations and vulnerabilities are not limited to one specific platform and ISA, but rather threatening the whole spectrum of computing hardware.

5) EVALUATION METHODOLOGY

The percent-wise distribution of different types of evaluation methodologies employed by the primary studies is shown in Figure 4(c). Overall, 55% and 22% of the primary studies opted to be evaluated on real hardware and FPGA systems, respectively. In contrast, 25% and 12% of the studies opted for simulation- and software implementation-based evaluations, respectively. And, the meager 2% are grouped under the other category, indicating that the other categories do not apply (e.g., the studies, which propose a criterion).

VI. DISCUSSION – ADDRESSING THE RESEARCH QUESTIONS

In this section, we address our research questions by using the primary studies in due length and scope. However, the gist of

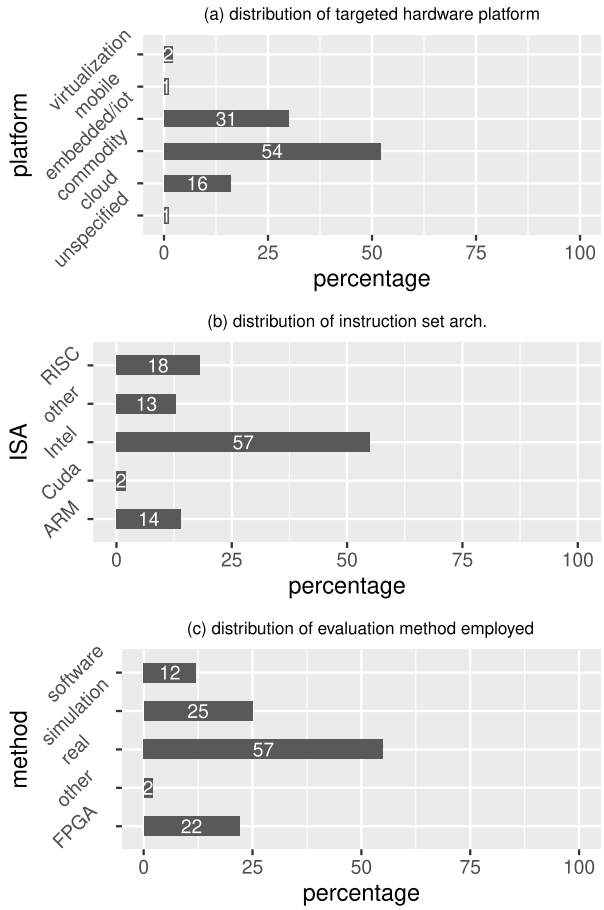

FIGURE 4. Percent-wise share of primary studies with respect to (a) target hardware platform and (b) instruction set architecture. The bar length indicates the percentage value and the integer value on each bar indicates the actual count.

following sub-sections span along the following lines: i) to-date almost all known microarchitectural components have been exploited in quite novel ways; ii) the motivations behind, reverse-engineering skills, knowledge of the microarchitectural intricacies at system-level, and availability of needed resources to research, prove to be sufficient in the hands of exploit developers; iii) upon public disclosure of an exploit, in due time researchers are able to come up with effective mitigations, albeit often at the cost of decrease in system performance; iv) long-term academic research is bearing fruit and has gathered sufficient knowledge that is improving and shaping security-oriented future of microarchitectures.

A. RQ1: WHAT ARE THE ASPECTS OF MICROARCHITECTURAL ARTEFACTS, WHICH CONTRIBUTE TO THE SENSITIVE INFORMATION LEAKAGE TO COMPROMISE SECURITY AND PRIVACY?

Undesired information leakage from the microarchitectures often originates due to the spatio-physical implementation of the CPU, the design optimizations, the trade-offs, and the temporal resource allocations among various functional units during the program executions. Effective resource allocation and optimal scheduling of the functional units can deliver the desired performance, but may fail to deliver perfect security. From the existing security research, the diverse aspects of the microarchitectures have been explored to discover the security vulnerabilities, which often involve one or more components being engineered into an information leakage state.

We identified a number of artefacts through our set of primary studies (Table 5) compromising microarchitectural security. Next, we provide a high-level overview, which should serve as a primer for a starter audience. The more focused and technically inclined security-specific account will be covered in Section VI-B.

Concurrent execution of multiple processes has an influence on the contents of shared and private caches in a multi-core environment. Furthermore, the hardware scheduling and resource allocation policy determines the assignment of cores to concurrently executing processes. Processes executing on the same core can access the core-private caches as well as the shared ones. However, processes even executing on different cores can indirectly influence the content of the private caches of other cores, through the propagation of the side-effects from the shared LLC cache to all cores [S13], [S18] Thus, code execution and data access, ripple through the cache hierarchy in ways, which an adversary can exploit through timing and access based channels.

The memory controller resolves memory accesses made by CPU onto the internal organization of DRAM, such as into channels, banks, rows, and columns. Furthermore, it also arbitrates and schedules multiple concurrent memory access requests in addition to overseeing the buffering of data for reads and writes according to the predetermined policies. The choice among different internal policies exhibits an effect on the timing variations and on the frequency of conflicts during bank- and row- address resolutions. Certain combinations of these policies give rise to exploitable timing differences, which an adversary can leverage [S21] to build a covert communication channel and utilize patterns in these timing differences to encode bits of secret message. In a recent work [S33], the authors described a feasible exploitation of memory deduplication mechanism to establish a covert channel in a cross-vm (virtual machine) scenario.

The buffer contents and the order of register allocation can significantly impact the internal state of the CPU core and could be exploited to engineer an internal state feasible for security exploitation. One such tactic is to prolong intervals of transient execution for certain processes, which would provide an opportunity to attacks reliant on transient state. It would also assist to increase the overall effectiveness of the similar attacks [S58]. Furthermore, under favorable internal states, potent electromagnetic and power side-channel attacks [S54] can be carried out. The internal state of a CPU has a direct correlation to power consumption and electromagnetic emissions. Analyzing these emissions can help a spy infer the degree of system activity as well as the specifics regarding the current workloads.

Intel SGX creates an isolated execution environment, called *enclave*, which provides a protected, secure, and isolated sandbox to code segments in a user process.

In these code segments, sensitive computations are carried out, which rely on some secret information, such as a cryptographic attestation. The notion here is to protect sensitive user processes from untrusted system software, such as an OS or a hypervisor. However, in recent years various microarchitectural attacks have been demonstrated to break the security provided by SGX, despite the strong isolation guarantees provided by this technology [S78], [S77]. In particular, attackers have been able to compromise authentication during the entry and exit protocol performed while delegating the computations from OS to enclave and vice versa. In another instance, malware has been able to conceal itself inside a secure enclave by abusing the SGX protection features, which enabled the malware to carry out attacks on neighboring enclaves [S77].

Modern microarchitectures enable speculative execution of programs. In a nutshell, processors can speculatively fetch and execute a stream of instructions along a predicted path ahead of time. Later, if the program actually executes along this predicted path, the pre-computed results are already available and ready to use. On the contrary, if the prediction fails, the thus-so-far speculative computation trail is simply discarded. However, research reveals that speculative execution leaves behind trails of temporary microarchitectural state changes, such as the cache lines being touched and/or evicted and the updates in the behavior of the branch predictors or prefetchers. The exploitation of aforementioned state changes led up to many side-channel attacks to exfiltrate sensitive information, which otherwise is not possible to do so [S71], [S8].

Some recent research revealed that prefetchers can be exploited to establish a high-bandwidth covert-channel across processes. A prefetcher has the ability to learn and retain patterns of memory accesses and leverage this knowledge to fetch data ahead of time. However, a malicious process can abuse this ability and selectively make the prefetcher to forget the strides it has learned for another process. This is typically feasible due to the limitation that only a finite number of patterns can be kept. Therefore, to make room for a new pattern, one of the old patterns has to be forgotten. Leveraging the aforementioned limitation, two spy processes can exchange secret messages by placing their stride patterns into the prefetcher and determining whether or not their strides are forgotten at a later time. Demonstrations reveal that attackers can build much more stealthier and high-bandwidth covert channels capable of transmitting information in the order of several tens of KBps with low error-rates [S38].

Modern CPUs are equipped with a range of onboard electronic modules to ensure reliability of the operations despite the changing conditions in the surrounding environment. Among other functionalities, these modules are meant to manage power, regulate voltages, and provide protection against excessive temperature. However, these modules are not devoid of side-channel leakage, through which an adversary can snoop into the inner workings of a CPU core. Typically, in adversarial settings, power and electromagnetic emanations are leveraged either to infer or to affect the inner state of a CPU core [S39].

In the preceding account, we observe that the artefacts leading up to the compromised security and privacy are not limited solely to one specific aspect of the microarchitectures, but typically span across the physical components, such as cache, memory, and registers. Nonetheless, they also span across the executional mechanics of the speculative execution and prefetching. Interestingly enough, the onboard electronics also contribute to the artefact emanations in power consumption and electromagnetic emissions.

B. RQ2: HOW SOME OF THE RECENT MICROARCHITECTURAL SURFACES WERE CRAFTED AND TURNED INTO FEASIBLE ATTACK CORRIDORS?

In RQ1 (Section VI-A), we presented a high-level overview of the artefacts emanating from the microarchitectural intricacies, which ultimately are leveraged by adversaries to compromise the security and the privacy of the computing systems. In RQ2, we intend to cover the ways, through which the aforementioned artefacts are utilized. To this end, we group the primary studies based on the leaking component and provide a brief account of how the attack corridors are actually created.

1) CACHES

From our set of primary studies (Table 5), microarchitectural cache-attacks compromising security and privacy, appear to be a frequent theme. In the following, we briefly summarize and highlight the aspects, through which this involvement was observed.

Fernando et al. [S70] audit the strength of the cache mapping functions, which map memory addresses to cache sets, through their proposed framework and reveal several vulnerabilities. The study emphasizes that sophisticated mapping functions, which obfuscate the address mapping to cache sets, are needed. The authors pointed out a malicious process can significantly infer the portions of a secret key by indirectly observing the cache sets being accessed during encryptions. However, this approach requires prior knowledge of how the addresses are mapped to the cache sets by the underlying hardware. The cache set mapping is typically performed through specific cache mapping functions, the details of which are often proprietary [30]. Weak cache mapping functions reveal a subset of the address bits that can be inferred by the malicious processes. In this regard, one-to-one mapping functions were found to be the weakest, which can reveal all of the bits of the set index. Moreover, larger memory footprints also leak a significant portion of the cache tag. Thus, information-theoretic security measures for the mapping function are absolutely essential.

Similarly, caches are found to be leaking ongoing data accesses of concurrently executing processes through fine-grained timing observations. In such a setting, a malicious process intentionally manipulates the cache contents to either transmit information to a co-collaborating process or to infer

secret data being accessed by a victim process. In such settings, cache access patterns of the spy process happen to conflict with in-cache data being processed by a victim process. Such conflicts lead to data evictions from the cache lines, followed by data loads for the affected process. These events of evictions followed by loads are observable in varying access latency, which can be captured through fine-grained time measurements made by a different spy process executing concurrently. To serve the malicious intentions, such time measurements can be leveraged to pinpoint the cache lines being in use by a victim process, which can then be associated with the different portions of the secret data, such as the secret keys. This information leakage through the timing channels is a prevalent threat as caches expose a large attack surface, which is difficult to mitigate. The timing aspect has been of particular focus in few of the primary studies [S67], [S15].

Saeshwar et al. [S36] demonstrate a feasible cache covert channel that, although, works on the principle of flushing a cache-line, does not depend on the native flush instruction (clflush) for this purpose. The authors point out that attacks relying on the flush instruction are known to be some of the fastest ones. However, the reliance on the flush instruction requires the sender and the receiver to be in tight synchronization for maintaining a low error rate. The authors conjuncture that higher data rates could be achieved, if reliable asynchronous coupling between the sender and the receiver could be established. To this end, they observe that sequentially accessing a large-enough address space (comparable to the size of LLC) implicitly triggers cache-thrashing operations where previously accessed entries from LLC are automatically evicted to create room for the new requests. Leveraging this property, a covert communication channel between a spy and a receiving process can be created. For this to work, a spy needs to consistently stay ahead and sequentially access the successive addresses in a large array. The receiving process, on the other hand, needs to slightly fall behind and observe the incurred LLC misses. The values of the bits in the communicated messages are inferred from these miss patterns. In reality, speculative execution and prefetchers can cause major disruptions by polluting the cache in the aforementioned scheme. To workaround this limitation, a coarse synchronization, such as once every few thousand accesses, is required between the sender and the receiver.

Thoma et al. [S102] argue on the effectiveness of employing randomization schemes as defences to a variety of cache side-channel attacks. To this end, the authors uncovered a new microarchitectural attack named Write+Write, capable of defeating cache randomization based defences. This attack leverages the Write-after-Write side-effects to establish covert-communication across CPU cores. These side-effects emerge whenever two physical addresses collide in a specific range during memory writes, in a sense when a write request is issued, a subsequent write operation to a nearby address in a specific 10-bit range is notably slower than any other address far out of this range. A predetermined set of addresses within aforementioned range are alternatively read and written such that, when write requests are issued among one of those addresses, a concurrent process reads and measure the time it takes for the rest of these addresses. If a pair of write requests do collide on physical addresses under aforementioned range, notable time difference is observed and vice-versa, this strategy is used to encode and secretly transmit ones and zeros of a covert message. A limitation of presented attack is its practicality stays only over short time periods and its stability is adversely affected by dynamic adjustments made to CPU clock frequency by power management hardware. Nonetheless, the attack demonstrates its usability and effectiveness to circumvent randomization based defense mechanisms.

2) MEMORY SUBSYSTEM

Semal et al. [S21] present two novel microarchitectural covert channel attacks, demonstrating a vulnerability surrounding the scheduler of the memory controller. These attacks are particularly potent in the context of cloud-based virtualization environments. The underpinnings of the attack dynamics involve a malicious pair of sender and receiver processes sharing regions of the DRAM banks. The sender process continuously creates some intentional memory-bank conflicts to encode bits of a covert message. These conflicts inadvertently also manifests into observable timing variations as the channel scheduler in memory controller experiences higher latency. The receiver retrieves the covert message through continuously performing uncached memory accesses in a pre-determined memory bank and observes the patterns of latency to infer upon transmitted bits of covert message.

Lindemann et al. [S33] reveal that the memory deduplication mechanism can be exploited by demonstrating that a spy virtual machine can identify the software configurations used by the co-located victim virtual machines sharing the same physical memory. The attacker first establishes the knowledge about the exact subset of the memory pages that would uniquely identify an application as well as its version. Such a subset of the memory pages is referred to as the application signature. The attacker overwrites the signature pages that are believed to be a part of the victim VM. The attacker silently waits for the deduplication to take place, then rewrites the signature pages, and finally measures the time required for these operations. The time measurements serve as an indicator of the pages that have been overwritten during the deduplication. This information is later used to determine whether the application of interest and its corresponding version is present in the victim VM. Similarly, Gulmezoglu et al. [S44] rely on the de-duplication mechanism as a prerequisite step to carry out yet another cross-VM attack on AES implementations.

Semal et al. [S64] describe an inter-process covert channel attack, named memory order buffer (MOB) attack. The attack underpins a side-effect of write-after-read hazard known as 4K-aliasing, which happens whenever the lower 12 bits of a virtual address match during a successive load and store operations. This effect inadvertently re-issues an additional

load operation, which causes the load/store bandwidth to drop momentarily, inducing higher latency. A spy process leverages this effect to encode the ones and zeros of a secret message by allocating a buffer at a page-aligned boundary. A value of one is encoded by filling this buffer. On the other side of the covert channel, a process monitors the trends in observed latency and recovers the encoded message bits accordingly.

Shi et al. [S7] advocate for reliance on general-purpose secure processors in trusted computing environments. Secure processors provide a tamper-proof trusted computing environment and protect information stored in regular memory devices by keeping the data in encrypted form. The authors audit a few contemporary secure processor designs and reveal their proneness to memory-fetch attacks. Although the data remains in an encrypted form inside memory, yet it is decrypted on-the-fly inside the processor during execution. However speculative execution poses a challenge as the instructions and data being fetched ahead of time are being decrypted on-the-fly and lead up to additional fetch requests in plain form on the memory bus. A spy snooping on the system bus can leverage these fetch patterns and selectively corrupt the bits of issued addresses to engineer alternate flows of instruction execution and/or data accesses in effort to disclose sensitive information. The authors propose remedial measures and alternate designs for the speculative pipeline, which incorporate the integrity verification of code and data.

Ravichandran et al. [S100] present a novel yet potent vulnerability, named Pacman. This vulnerability is among first of its class where a pair of individual vulnerabilities are teamed up to break existing security barriers in-place to curate a more potent attack. Interestingly, pacman targets Apple's M1 ARM CPU and defeats an important memory protection mechanism based on pointer authentication. ARM introduced pointer authentication to protect pointer integrity as a security feature, and has been in wide use on variety of systems ever since. A memory pointer is protected through storing a computed hash value alongside its contents. The hash value serves to establish if there has been an unwanted pointer modification performed, presumably by an attacker. Whenever, a pointer is used its integrity is verified by validating aforementioned hash value. An attacker who wants to modify a pointer has to correctly infer the correct hash value after a modification to prevent the system from detecting a pointer tampering. Naive brute force attempts of attacker to infer correct hash value will not work as it would lead to program crash once the system detects a hash mismatch during pointer authentication. However, the attacker synthesizes an oracle which he could utilize during speculative execution phase, alongside an experimental guess of a hash value, and queries the oracle about legitimacy of said value. Upon a correct guess the attacker can proceed with pointer modification and its associated hash value, otherwise the attacker would wait for subsequent speculative phase with a newly guessed hash value. The authors describe a practical and systematic way to craft aforementioned oracle and guess for the hash value. To this end, pacman successfully demonstrated all experimental scenarios of overcoming pointer authentication mechanism and hijacked pointers to execute malicious payloads performing attacks from userspace.

Zang et al. [S103] outline that data-execution prevention, and execute only memory protection are effective security hardening mechanisms to protect critical programs and defend against attacks employing principles of return-oriented programming(ROP) [31]. The study makes an important observation that system call routines lack a subsequent return instruction which is deemed essential for any ROP technique, however, exploiting the signal handler mechanism can circumvent this lack of return instructions. Upon exploitation these system calls can be utilized to assemble a set of ROP gadgets through which a broad range of exploitations can be carried out. However, for presented approach to succeed, syscalls need to be timely interrupted during their execution through signals such as illegal memory access. Carefully crafted illegal memory accesses when carried out, open up an avenue where code segments are stitched together through a set of system calls, to craft a complete microarchitecture attack. The authors demonstrate their approach by successfully carrying out three different scenarios of creating a back-door into a victim system.

3) MICROARCHITECTURAL BUFFERS

Barenghi et al. [S59] investigate the extent of the microarchitectural information leakage attributed to the pipeline buffers. They demonstrate that significant information leakage is observed in the power traces obtained during the compute-intensive program executions. The observed leakage turns out to be correlated with the order, in which the registers are allocated for the memory loads and stores, rather than the data dependencies among the instructions. They also observe that minor changes in the order of register allocation can potentially lead to exploitable vulnerabilities. In particular, inter-stage buffers among the issue and execution stage (IS/EX), execution and writeback (EX/WB) stage, ALU output, and memory data registers are the main contributors to the said leakage, which tends to get highly influenced by the order of register allocation. In [S54], the authors also demonstrate the role of the pipeline buffers and the associated functional units in causing power-based information leakage.

Schluter et al. [S71] focus on the role of the short-time buffers in information leakage, which are present on CPU cores. In particular, they investigate the role of the line-fill buffers (LFBs) in harboring microarchitectural data sampling attacks. During speculative execution in the event of page faults, such attacks exfiltrate transient data from concurrent processes executing on the same logical cores. Moreover, the study cites the role of the store buffers, fill buffers, and bus configuration registers, which are prone to leakage.

Kim et al. [S58] present a more potent variant of Meltdown [1] attack by using the return stack buffers (RSBs) to widen the window of transient execution. The proposed exploitation technique enables the establishment of a covert

channel without requiring a context switch, providing better tolerance to noise-based countermeasures.

4) TRUSTED EXECUTION ENVIRONMENTS (TEE)

A notable number of primary studies have focused on the vulnerabilities of TEE. Next, we briefly summarize their aspects of exploitation.

Gysenlik et al. [S82] demonstrate, in a novel attack, that legacy features for backward compatibility of x86 instruction set can be leveraged to compromise the security of 32-bit SGX enclave. The attack abuses x86 segmentation unit to reveal enclave memory accesses at the granularity of page level and, in more favorable conditions, even at the byte level. In essence, the proposed attack loads the segmentation unit registers with an engineered configuration, which ultimately causes either a general protection fault or an ordinary page fault. The pattern of page faults reveals the secret-information-dependent memory accesses inside the enclave. From this information, the structure of the sensitive code doing secret processing is inferred and the control flow can be spied upon. This study is notable in the regard to be the first study to expose avenues for newer attacks exploiting the legacy x86 features, which would remain a part of the Intel CPUs providing backward compatibility in the foreseeable future.

Moghimi et al. [S78] present an attack to retrieve secret information from a SGX enclave. The proposed attack exploits the false dependence of the memory read-after-write operations during serialized accesses of specific 4K-byte memory blocks by a victim process. The victim, which has constant-time code implementation for additional security, performs cryptographic operations inside the enclave and performs frequent enclave-memory operations. The aforementioned serialization causes observable and distinguishable latency patterns, which a spy process ultimately harvests to infer upon the bytes of a secret cryptographic key.

Schwarz et al. [S77] present a practical SGX enclave-based malware, which operates from a compromised enclave. The malicious enclave exploits the SGX enclave protection features to conceal itself from the operating system thus, thwarting the discovery while remaining stealthy. And, the attack can be carried out either against the other co-located enclaves or against the secure docker containers co-located at the same machine. The underpinnings of this attack utilize the Prime+Probe [32] technique to determine the cache-access patterns by observing the memory access latencies, such that an RSA key processed by a victim process can be discovered.

Skarlatos et al. [S32] target the privacy of SGX enclaves through microarchitecture replay attacks. Such attacks rely on hardware support to roll back and re-execute instructions under certain preconditions in a quasi-transient state. The attack details a SGX spy, which exfiltrate secret information from the enclave-private memory of a victim by making it repeatedly replay on page faults. The repeated replays enable the adversary to break the privacy of the enclave and figure out the secret enclave data to the extent possible. The proposed approach positions itself as a potent technique, which can function effectively even in the presence of considerable noise.

Lang et al. [S72] point out the tactics of a malicious adversary having complete control over OS, through which the traffic to and from the outside world to a secure enclave can be influenced. One way is to frequently interrupt and preempt the execution inside an enclave from outside. Such repeated interruptions could potentially cause an enclave to move into some meta states, through which the secret information can be exfiltrated. Moreover, in such a setting, the adversary, being an untrusted OS, can block, delay, replay, and modify all the communications issued from outside the enclave, causing the establishment of a weakened inner state, against which an attack can be carried out.

Han et al. [S23] introduce a covert-channel attack that works across Intel SGX enclaves. The attack exploits a special cache, which is part of the memory encryption engine (MEE) of SGX hardware. The authors observe that the aforementioned cache keeps a portion of the integrity tree storing some enclave-private data. And, it turns out that the number of MEE cache accesses performed during the fetching of this enclave-private data, is directly influenced by the internal state of the tree. The proposed attack follows the footsteps of a typical Prime+Probe attack and systematically forces the integrity tree updates by influencing the MEE cache. The patterns observed through the forced updates are then leveraged to encode and transmit the bits of a secret message across the enclaves.

5) PREFETCHERS

Up until the recent past [5], the microarchitectural attacks targeting the prefetchers in a practical setting were non-existent. Recently, researchers have exploited the prefetchers after the empirical investigation of their inner workings and leveraged the knowledge gained to craft successful attacks. Next we provide a brief account of these attacks.

Patrick et al. [S38] present a microarchitectural covert channel attack targeting the hardware prefetchers on modern Intel CPUs. The presented attack establishes a bi-directional, high-bandwidth covert channel, which is stealthier and can avoid detection. The attack employs a stride prefetcher to differentiate between accesses to the data blocks with the ultimate goal of figuring out whether the blocks are retrieved from the memory or have been already prefetched into the cache. More specifically, the spy process, which concurrently runs with the victim process, intentionally engineers some sequences of prime and evict operations from the L3 cache to perturb the thus-so-far learned sequences by stride prefetchers. The degree of the perturbation is used to encode the bits of a covert message, which is consequently decoded by the receiving spy process through the monitoring of prefetching behavior.

Note that the prefetchers are used to build a covert channel in the aforementioned study [S38]. Interestingly enough, in the case of cache side-channel attacks, prefetching unintentionally hinders the effectiveness of the attacks. Since the prefetching mechanism speculatively brings data into

the cache, it weakens the attacker's ability to distinguish whether a cache line have been fetched on demand by the victim or have been speculatively brought by the prefetcher for the victim. Wang et al. [S46] point out this shortcoming and present a work-around solution in order to enhance the effectiveness of the cache side-channel attacks, thus making them more potent. The main challenge they address is to understand and reliably model the uncertainty in the prefetching patterns, which originates due to the undisclosed proprietary details of the inner workings of the prefetchers. In the aforementioned work, the authors reverse engineer the inner workings of the Intel CPUs and develop a statistical description of the prefetching mechanism implemented by them. This description is later leveraged to strategically craft and place probes to build the enhanced versions of the cache side-channel attacks. Furthermore, they demonstrate a Flush+Reload [33] attack, which is more potent and equipped with the ability to effectively operate under the disruptive behavior of prefetcher.

Ibrahim et al. [S97] argue that sophisticated interactions among various microarchitecture components give rise to potent and often enable previously undiscovered attacks from userspace. Effective defense and offense often require detailed knowledge of inner-workings of aforementioned components however, such details are usually unpublished for many CPUs. The authors introduce concept of leakage templates, to abstractly describe and identify specific dynamics of known attacks. The presented approach, leverages these abstractions to discover leakage-causing code segments in a binary and once discovered, variants of culprit code segments are synthesized through instruction fuzzing, operand mutation and contextual analysis. Later these synthesized variants are executed on a microarchitecture of interest and changes in the microarchitectural state are observed. Among the observed state space, vulnerable states are identified and the code variant causing it. Equipped with this knowledge, practical exploits for a given microarchitecture are crafted. The approach showcases its strength by uncovering a novel cache eviction and a prefetcher based exploits on an ARM CPU.

6) PCIe

Tan et al. [S53] present some attack scenarios by analyzing the patterns in PCIe contention where a spy process can exfiltrate sensitive information spanning from keystroke timings to figuring out the webpages being visited and the machine learning models being used. Note that these attacks are particularly relevant in the settings of cloud computing and data centers where co-resident virtual machines can snoop on each other. The underpinnings of the proposed attack assume a pair of peripheral devices connected to the same PCIe switch, one of which serves under the spy process while the other serves the victim process. The spy is interested in learning the distinguishable patterns in IO latency routed through the PCIe switch. The patterns are later post-processed through a supervised learning approach to infer the victim's state. The spy can intentionally choke the PCIe switch to a degree that causes these patterns to emerge as a side-effect to the intermediate buffering of reliable data transfers.

7) PERFORMANCE COUNTERS AND GPUs

Graphical Processing Units (GPUs) became one of the must-have components of modern computer systems, which are meant to provide enhanced capabilities and performance for graphical workloads. However, besides graphics processing on modern systems, they can be employed to process intensive workloads such as scientific computation and machine learning applications. The flexibility of the GPUs for the aforementioned purposes is achieved through leveraging the GPU APIs (application programming interfaces). The internal registers of GPU capable of monitoring and profiling executions, are also made available in these APIs. Although a GPU can serve multiple workloads through time sharing, a workload with malicious intentions can abuse the microarchitectural components of a GPU to establish a side-channel.

Naghibijouybari et al. [S35] demonstrate the information leakage through the GPU performance counters to a spy process by inferring the graphics workload being rendered for a victim process, which was enough to fingerprint the websites. In the proposed approach, a victim process, such as a web browser, renders the web graphics through the GPU, which leaves a trail of the deviated values in the performance counters. The spy process employs machine learning to classify these trails, such that the websites being visited can be identified with high accuracy. In a similar setting, the interleavings between the spy, victim, and other processes due to the GPU scheduling, are shown to influence the values of the performance counters, which, in turn, enables the spy process to infer the scheduling order of the workloads. The authors also present some similar attacks operating by abusing the low-level GPU APIs, which they discovered from the accompanying GPU-programming SDK (software development kit). These abused APIs enable one to read the values of the performance counters as well as the contents of the internally used GPU-memory allocation registers. Their attacks showcase the exfiltration of keystroke timings, website fingerprinting, and the discovery of the parameters of a neural network workload.

Dutta et al. [S13] demonstrate the feasibility of two microarchitectural covert channel attacks from GPU to CPU and vice versa. The authors showcase two such attacks, on an Intel CPU with onboard integrated GPU. This particular design share components such as the last level cache (LLC) and ring-bus interconnect among GPU and CPU. A spy could utilize latency on the aforementioned shared components and establish a covert channel. In the first attack, the spy employs shared LLC and uses Prime+Probe [32] technique to encode and transmit bits. The spy on the sender side, primes the LLC cache set, which is probed by the spy on the receiver side. If the sender wants to send zero, it does not prime the cache set, but the receiver still probes it and deduces this transmitted zero. In the second attack, the sender and the receiver simultaneously cause contention, which is higher

than the usual amount on the ring-bus to encode a zero. Furthermore, the authors argue that the GPU-CPU cross-component attacks are novel and could deliver an attacker newer stealth capabilities and that any research aimed to develop countermeasures in these contexts will face unique challenges.

8) ONBOARD ELECTRONICS

Sehatbakhsh et al. [S39] demonstrate a microarchitectural security vulnerability by targeting the onboard electronics of the CPU cores. They, in particular, focus on the voltage regulator module (VRM), which serves to stabilize the voltage variations arriving at the CPU cores in the presence of power fluctuations or workload variations. A CPU can also switch to one of the suitable operating power states via the software configuration of VRM. The authors observe that electromagnetic emanations from VRM are directly correlated with the operating state of the CPU. Leveraging this behavior, a spy could establish a covert communication channel through VRM manipulations by switching among possible operating states and hence could transmit secret information from inside the victim to a remote system wirelessly over the air. The authors showcase keystroke logging through this proposed approach.

Schwarz et al. [S79] present NetSpectre, a variant of Spectre attack [2] capable of carrying out remote attacks across the machines over a network. The authors state that unlike the traditional Spectre attack, which employs a cache-covert channel and leverages the observed latencies during the cache accesses, their presented attack utilizes the AVX2 instructions and their effects on the observed latencies under a special power-saving mode on CPU. The authors report that CPU can power up the upper half of the AVX2 unit whenever an instruction requires 256-bit computation. Otherwise, this unit is kept powered down. However, this powering up on the need basis manifests as observable latency in AVX2 instruction execution, which is more noticeable during operations involving concurrent network accesses. Secret data can covertly be exchanged between a pair of remote machines, such that the spy machine runs a compute function using the AVX2 instructions, receiving the input from a different machine over the network and returning the result back to the requesting machine. Another spy process running on the requesting machine can observe the timing differences in relation to the input values, thus can infer the bits of the covertly sent message.

Liu et al. [S99] highlight that modern CPUs dynamically adjust their clock frequency and operational voltage to achieve energy savings and optimize operations in lieu to varying workloads. The onboard power management architecture continuously monitors and reactively adjusts CPU frequency to keep operating within safety limits. However, these dynamic adjustments by power management system leads up to a timing-based side-channel through which an attacker could exfiltrate sensitive information. In a nutshell, the attacker profiles a constant-time implementation of a victim workload under known conditions and presets a limited power management policy to record any frequency adjustments being made while victim executes. After ample observations, the attacker launches a stress-testing workload with carefully crafted input parameters, under attacker's control. The attacker intend to stress-out current power policy such that it would lead to reactive frequency readjustments as CPU demands for more power. In this state, the attacker makes ample further observations so he can correlate changes in current frequency adjustments and contrast against its previous observations. With large number of observations, attacker is able to statistically infer upon the data being processed by victim workload. Under aforementioned strategy the study showcases successful retrieval of cryptographic keys from a constant-time implementation of a victim workload. Furthermore, this attack is both novel and potent in the sense that attacker does not need any special privileges and can operate through userspace and work across many mainstream CPUs. Dipta et al. [S91] outline a similar attack abusing direct read access to operating frequency values in userspace. The authors argue that instantaneous values of operating frequency are directly correlated to degree of system utilization. A passive userspace attacker can silently monitor and observe the trends in frequency changes, the presented approach employs statistical analysis to fingerprint websites being visited by a user. Furthermore trained machine learning models were used to analyze this data and user keystrokes were inferred. Fendri et al. [S93] presents an interesting attack, through which an attacker can infer individual instructions while they are executing thus, compromising code confidentiality and ultimately recovers sensitive information. The presented attack is referred as disassembly-through-side-channel and is particularly effective in the context of embedded/IoT devices. The attacker first obtains a design netlist of target CPU to simulate the power consumption within its various subsystems, under various scenarios in relation to pre-specified instruction streams thus, develops a behavioral profile. Next, a multi-layer machine learning classifier is trained against said profiles of target CPU in conjunction to CPU-specific features. Once trained, the classifier is leveraged to disassemble instruction streams executing on a victim CPU on a target system, through its captured power-consumption traces. The study showcases practical strengths of proposed approach to be able to successfully disassemble, about 96% of instructions being executed on two different target RISC-V CPUs.

9) SPECULATIVE EXECUTION

Chowdurry et al. [S8] observe that the conditional branch executions in a speculative path have an effect on the continuously maintained history of the branch predictors. Regardless of whether the speculation ultimately leads to an abortion or not, the branch history is never restored back to its original state. Thus, the internal state of the branch predictor is permanently affected. An adversary can either passively observe or actively modulate the speculatively affected history as a means of covert communication with a malicious peer process. The information encoding is carried

out by using the predicted outcome of a conditional branch and/or the predicted target address.

Last but not least, Trippel et al. [S9] leverage formal methodology to discover vulnerabilities under speculative execution and synthesize the corresponding exploit programs. Given a specification of the microarchitecture represented as a special graph, the approach reasons about the orderings and the interleavings of the hardware execution events in relation to the program executions. More specifically, the indicative behaviors leading up to the exploit scenarios are mined. Once an exploitable pattern is found, several programs can be synthesized in an automated manner, exploiting the vulnerability in some potentially feasible ways. The proposed approach showcases its unrivaled ability by discovering several novel and existing vulnerabilities regarding pure speculative execution. Moreover, notable timing- and cache-based vulnerabilities have been uncovered through the same approach.

C. RQ3: HOW EFFECTIVE ARE THE PROPOSED COUNTERMEASURES OF MICROARCHITECTURE SIDE-CHANNEL ATTACKS AND WHETHER THESE COUNTERMEASURES ARE GENERALIZABLE? CAN THESE GENERALIZED COUNTERMEASURES PREDICT/ PREVENT ZERO-DAY ATTACKS?

We identified the studies from our list of primary studies (Table 5), in which the main focus has been presenting a *countermeasure* approach to address a microarchitectural threat. We observed that the primary studies discussed the scope and the effectiveness of the proposed approaches to various degrees. To answer our current research question, we, therefore, chose to include a subset of those countermeasure-presenting studies, in which the effectiveness and the incurred overhead of the proposed approaches had been evaluated.

We, furthermore, observed that some primary studies, in conjunction with numerical reporting, also qualitatively state in adjectives the degree of the effectiveness and the performance overhead of their approaches, such as *highly effective approach* and *negligible performance overhead*. On the contrary, other studies only relied on numerical reporting. This inconsistent reporting style forced us to follow the following categorization scheme, which we utilized to group the studies to aid in answering RQ3. More specifically, we opted to categorize the reported *effectiveness* of the countermeasures in two levels: *highly-effective* (H_C) and *effective* (E_C). Similarly, we categorized the *performance-overhead* in three levels: *low* (L_o), *moderate* (M_o), and *high* (H_o). We assigned H_C label to those studies where the authors described the effectiveness of their approach in superlative adjectives, such as *quite-*, *highly-*, and *extremely-*, or in numerical reporting between 85% and 100%. Otherwise, the E_C label was used. Similarly, the performance-overhead reporting under the adjectives, such as *negligible-*, *low-*, and *practically-zero*, or numerical reporting between 0% and 5% has been assigned L_o. The studies using the comparative degrees of adjectives, such as *some-*, *tuneable-*, and *moderate-*, or numerical reporting between 6% and 20% were assigned M_o. And, for the remaining studies, the H_o label was used.

TABLE 3. Categorization of primary studies which cover countermeasure approaches.

Study ID	Effective	Overhead	Zeroday
S40	H_c	L_o	
S48	H_c	L_o	
S6	H_c	L_o	✓
S12	H_c	L_o	
S67	H_c	L_o	
S101	H_c	L_o	
S31	H_c	M_o	
S15	H_c	M_o	✓
S41	H_c	M_o	
S45	H_c	M_o	
S51	H_c	M_o	
S43	H_c	M_o	
S72	H_c	H_o	✓
S75	E_c	L_o	
S56	E_c	M_o	
S27	E_c	L_o	
S52	E_c	L_o	
S47	E_c	L_o	✓
S98	E_c	L_o	
S24	E_c	H_o	
S86	E_c	H_o	
S88	E_c	H_o	
S87	E_c	H_o	
S89	E_c	M_o	
S90	E_c	M_o	
S85	E_c	M_o	
S55	E_c	M_o	

Table 3 presents the categorization results we obtained for our primary studies. Next, we discuss the answers obtained for the individual parts of the current research question.

1) EFFECTIVENESS AND GENERALIZABILITY OF PROPOSED COUNTERMEASURES

The studies selected to answer this research question (Table 3) comprise 25% (27 out of 104) of our list of primary studies (Table 5). Regarding the effectiveness, 48% (13 out of 27) of the studies are classified as highly-effective and the remaining 52% are classified as effective. Regarding the incurred overhead, 40% (11 out of 27) of the studies report low, 40% report moderate, and the remaining 5% report high overheads. Ideally, a countermeasure demonstrating high effectiveness with negligible overhead is desirable, which may, however, not always be feasible. Next, we briefly outline the proposed countermeasures from the individual studies given in Table 3.

Fang et al. [S67] present a countermeasure addressing the cache-based covert timing channels, which tend to be an essential vehicle in a large number of microarchitectural attacks. The proposed approach selectively monitors the specific regions of the cache memory exhibiting some suspicious behavior. The cache-miss patterns of the regions of interest are recorded and constantly analyzed. The size of the monitored regions can also be adaptively increased or

decreased as needed. Any statistically significant deviations from the predetermined normal behavior raise alarms.

Panda et al. [S12] present an approach to defend against the cache eviction attacks employing time measurements. These attacks open up avenues for cross-core attacks and have been prominent in recent years. The proposed approach leverages the automatic activation of a specific prefetching mechanism, known as back-invalidation-hits-triggered prefetching (BITP), to fill the L2 and LLC cache. In this context, *back invalidation hit* refers to the phenomenon when a block from a lower-level cache, such as LLC, needs to be evicted, which also happens to be present in a higher-level cache, such as L2, thus needs to be evicted as per cache coherence policy. As eviction takes place BITP prefetching kicks in and loads a new block from the main memory into the cache hierarchy. The proposed approach intentionally triggers BITP prefetching frequently, which inadvertently interferes with the time measurements being carried by a spy, thus introducing imprecision in those time readings and hampering the success of an ongoing attack.

Li et al. [S98] emphasize on mitigating cache attacks carried out under general and speculative execution. The study outlines that a typical cache attack has three essential phases and hindering at-least one of the phases effectively disrupts an ongoing attack. The three phases broadly span across: an initial state of cache that an attacker can leverage, victim's access of cache accompanied by a change in cache state, and attacker determines the cache state change typically through timing to infer victim's secret. To this end the study proposed design of a secure and performant prefetcher which would interfere with one or more of the aforementioned attack stages. At its core, the proposed prefetcher accurately predicts the cache-lines which a victim during execution would evict, followed by prediction of cache access patterns by an attacker to perform its timing measurements. Guided by these predictions the proposed prefetcher strives to obfuscates the attacker by aggressively pre-loading a set of cache lines of interest, before attacker gets a chance to make a meaningful measurement. The experimental results validate the effective defense with very low overhead.

Yuce et al. [S6] address fault attacks where the adversary engineers a controlled fault in the microprocessor through the careful manipulation of the operating conditions, such that the execution gets into a vulnerable state, through which the secret information can be exfiltrated. These attacks are particularly relevant in embedded and IoT systems. The proposed countermeasure relies on providing a secure exception mode for a microprocessor by using a microarchitectural extension together with a trap mechanism in software. Furthermore, upon detecting an injected fault, the hardware checkpoints are utilized to bring the execution to a good state. Checkpointing is a mechanism, through which the sanity of the execution is periodically verified by keeping a trail of surpassed milestones termed as *checkpoints*. During checkpointing, if a fault in the execution trail is detected, the ongoing computation is discarded and the execution is resumed back from the last known good checkpoint. Specifically, the proposed approach maintains a hardware checkpoint of the critical system state periodically. In the event of a fault injection attempt, the checkpoint is frozen, a secure trap handler is initiated in the software, and the user space is notified. The user space can then optionally specify the recovery policy for the detected fault and ultimately roll back to a known good state, thus thwarting the attacker's attempt.

Kumar et al. [S40] propose an approach to enhance the security of the crypto-processors against the power and electromagnetic emissions, which could potentially be harvested remotely to compromise security. The study argues that the contemporary approaches solely address the attacks that aim to infer the cryptographic keys through the time-domain analysis of the harvested emissions, yet they remain prone to the frequency-domain analysis. Note that the time- and frequency-domain analyses aim to quantitatively study the signal properties, such as the time-varying behavior in relation to its frequency components. The proposed countermeasure involves a hardware extension in the low-dropout regulator (LDO) module of the crypto-processor, utilizing controlled randomization within its control loop. This, in effect, provides resistance in the time and frequency domain, enabling the AES and RSA modules of the crypto-processors to remarkably deter the aforementioned attacks.

Mane et al. [S48] propose an approach to systematically eliminate the exploitable side-channel leakage to enhance the resilience of the block ciphers, such as AES and DES, in embedded processors. The study argues that the roots of the aforementioned leakage stem from the data-dependent electromagnetic signal transitions, which manifest themselves from data-dependent processing at the microarchitectural level. The study observes that the aforementioned leakage can effectively be mitigated at the level of implementation with the employment of dual-rail pre-charge logic (DPL). DPL can be materialized by simultaneously storing and processing every data bit and logic operation. The study prototypes and evaluates a soft-core CPU synthesized on an FPGA board, with a custom instruction set and an optimized memory organization scheme under the guidelines of DPA, demonstrating promising results.

Zhang et al. [S31] present a countermeasure to harden the Intel SGX enclaves, such that information leakage caused by the engineered page faults can be prevented. In these exploitation attempts, the adversary relies on the deterministic memory access patterns against a known input. The proposed countermeasure is based on emulating a secure memory subsystem leveraging an enhanced ORAM (oblivious RAM [34]) protocol to load code and data into a pair of virtual caches. These caches periodically shuffle and re-organize their contents, which, in effect, poses a barrier for the adversary leveraging the engineered page faults. Furthermore, the proposed approach exhibits a tuneable performance overhead, thus enabling the system policy to balance the trade-off between security and performance.

Guo et al. [S15] present a countermeasure to the cache-timing attacks in the context of speculative execution

where intermediate transient states occasionally lead to vulnerable cache states, from which the secret information is exfiltrated by analyzing the patterns in the access latencies. The proposed approach relies on symbolic execution to systematically explore the state space of a program at conditional branches during speculative execution, such that the side effects on potential paths can reliably be analyzed. More specifically, the side effects of each path of interest on the cache memory are accumulated to create the leak predicates, which subsequently are utilized to perform cache behavior analysis using constraint solving. Paths leading up to sensitive states, which can potentially leak information, are avoided through tailoring the program executions.

Brotzman et al. [S86] present an approach to countermeasure cache attacks which exploit secret dependent leftover execution footprints under speculative execution. The study argues that speculation-aware static code analysis combined with precise cache models remain insufficient to screen vulnerable code segments. To this end, under the light of known cache models and recent attack strategies, this work proposes speculation-aware program semantics and a security definition. The proposed semantics leverage a prediction oracle which returns predicted outcome of a branch, and architecture dependent length of speculative transactions. Given this information the approach reasons about sequence of emitted microarchitecture events and tracks memory locations being accessed while instructions being speculatively executed thus, identifies the scenarios which would give rise to information leakage. Existing tools for auditing microarchitecture security could be strengthened with proposed approach and would detect vulnerable code segments which remained undetected previously.

Sakalis et al. [S101] discuss microarchitecture replay attacks, abusing speculative execution can trap execution of victim application in a loop and could perpetually amplify the attack by executing it indefinitely, regardless, whether the software has already been hardened against replays. To this end, the study presents a hardware only defense, named delay-on-squash, that tracks squashed instructions and prevents them from being replayed under subsequent speculative executions, thereby achieving remarkable security against these attacks with moderate hardware and execution overheads.

Vougioukas et al. [S41] focus on mitigating the exploits relying on the hardware branch predictors. Although the branch predictors are crucial for a high-performing system, they have been a part of the security exploitations where deliberate context switches influence the branch predictions. As a result, an adversary can alter/perturb the instruction flow of a context, which is unrelated to the current execution context, thus opening up an exploitable state. The proposed countermeasure uses a branch retention buffer as a novel mechanism to maintain isolated contexts during context switches, thus preventing vulnerable history alteration in the presence of malicious attempts.

Kiriansky et al. [S45] present an approach, which is based on minimal hardware modification to defend against a broad class of cache-based attacks, involving the ones relying on timing and speculative execution. The proposed approach utilizes the strong isolation achieved through protection domains. These domains segregate the cache hit/miss metadata information, line replacement data, and cache update policies. Based on the security requirements, the domains can be either adaptively granularized at the cost of the performance or nested together to establish a more performant, yet coarse-grained policy. The authors also argue that the proposed countermeasure is inherently robust against speculative execution attacks, such as Spectre.

Chouary et al. [S88] outline that during speculative execution a set of instructions, termed transmitters, inadvertently could aid an attacker to build a microarchitectural covert channel. Transmitter instructions which typically are branches and loads, can mis-speculatively execute with secret operands, which they would not execute with such operands under valid executions. The study conjectures that it is safe to delay, rather prevent transmitter instructions, iff it can be proven that secret operands were already leaked by a prior non-speculative execution. Based on proposed conjecture the study develops a speculative privacy tracking hardware protection, which delays execution of transmitter instructions until it could be proved that corresponding operands leak during non-speculative program execution. Once such a secret operand leak is detected then it is taint-tracked and guarded through fence instruction to avoid a subsequent exploitation. The study showcased the effectiveness of proposed approach albeit with non-trivial overheads.

He et al. [S51] propose a number of mitigation strategies for flush-based cache attacks in the context of ARM-based embedded/IoT systems. The presented work highlights the potential security threats demonstrated by the Spectre attack [2], which leverages the cache-line flush instruction, the abuse of which leads up to the exfiltration of sensitive information during speculative execution. The proposed countermeasure provides a secure cache-line flush operation that encapsulates the native flush instruction together with an accompanying mechanism to monitor its invocations. The invocations are monitored constantly for any sequence of suspiciously close invocations to detect and pinpoint the culprit process, which if found to be malicious, is blocked to safeguard the system.

Buiras et al. [S87] emphasize that observational models play a vital role in the security analysis of information flows. The authors introduce observation refinement technique to guide exploration of state space with a focus on hardware components of interest. To this end, the authors extended an existing model validation framework incorporating proposed model refinements through which, a new speculative leakage vector in ARMv8 architecture is uncovered. Last but not least, proposed approach also discovered a new vulnerability SiSCLock arising, during speculative execution of ARM Cortex-A53 processor.

Pham et al. [S43] present a countermeasure against the timing attacks in the context of embedded/IoT systems. The proposed countermeasure, which is based on program

diversification, uses a tailored compiler to generate custom instructions for the security-sensitive program regions. The aforementioned instructions exhibit diverse timing characteristics every time they are executed. However, hardware support is required for such custom instructions. The proposed countermeasure is evaluated by synthesizing a soft-core CPU on an FPGA board, delivering a mitigation performance of 86%.

Lang et al. [S72] present an approach to mitigate access/trace-driven cache attacks on secure Intel SGX enclaves. The mitigative approach leverages multiple concurrent threads of execution inside an enclave to monopolize the whole CPU during the security-critical computations. These threads perform either some enclave-related computations or some dummy computations to keep all the cores occupied, as a result of which, any potential adversary is starved due to the unavailability of the CPU. It is shown that although the proposed countermeasure is effective, it imposes considerable performance overheads.

Wu et al. [S55] highlight the role of Simultaneous Multithreading (SMT) in contention-based attacks. They argue that SMT provides a broader surface for resource contention in microarchitectural components per logical core than across physical cores. To strengthen the security posture against the aforementioned attacks, the proposed approach limits SMT on logical cores and the threads performing sensitive computations are exclusively and separately scheduled on available physical cores. To this end, a tailored user-level thread library is provided, which ensures the thread placement on the dedicated cores, limiting an attacker's ability to target the shared CPU resources on the victim's core under a contention-based side-channel.

Payer et al. [S75] present an effective proof-of-concept countermeasure system for a wide range of microarchitectural attacks, which is based on the concept of intrusion detection. The proposed system employs a plugin-based architecture, which can be equipped by developing an attack-specific plugin. The whole system leverages the statistics collected from the hardware performance counters on a per program basis to monitor the deviations from the normal behavior profiles. On a system-wide level, the profiles from multiple processes are correlated to determine whether multiple processes are involved in an ongoing attack. The proposed approach is showcased by effectively detecting the row-hammer, CAIN, and other cache-based attacks with acceptable performance overheads.

Busi et al. [S56] layout some security-centric design guidelines for the architectural extensions with regard to the isolation mechanisms, such as enclaved execution. The study particularly targets the small microprocessors due to their widespread utilization in the embedded/IoT systems and argues that enclave-isolation mechanisms are relevant more than ever in this domain. However, the contemporary designs are found to be exploit-prone in a number of ways, particularly to the fault- and untimely interrupt-based attacks. To this end, the presented approach lays out the formal criteria for any security-related microarchitectural design extension to ensure strong security posture. In particular, the study demonstrates a security-hardened enclave design, which is provably secure and resilient to the fault- and interrupt-attacks.

Chen et al. [S47] propose an approach to address the compromised security of Intel SGX enclaves through untimely interruptions and during speculative execution. The study points out that Intel has been unable to fully mitigate and fix the security vulnerabilities of these SGX enclaves. Therefore, they will remain exploitable in the foreseeable future. At a very high level, the proposed approach relies on temporarily disabling the enclave interrupts while carrying out sensitive computations. Furthermore, some constrained execution conditions are proposed for the operating systems to ensure sufficient protection against the attacks reliant on speculative execution. Lastly, a compiler-assisted tool is proposed to protect the sensitive code executing inside an enclave by embedding code regions performing attestation, sealing, and unsealing of the secrets required.

Cook et al. [S89] discover a shortcoming when it comes to employing machine-learning (ML) based approaches to countermeasure and analyze microarchitecture attacks. The authors elaborate that ML at its core is a black-box approach and often lead to largely incorrect interpretations of actual mechanics of attack under consideration. Such is the case in one of recently published sweep-counting-attack [35] where, a neural network based analysis on cache-traces led to the false notion that caches were the actual culprit, and were main sources of leakage. However, this later found out to be not true when, more traditional microarchitecture analysis methodologies are applied where, it found out that untimely system interrupts are the actual culprit. Furthermore, the study builds the case that interrupt-based attack vectors have the potential to mount powerful attacks from unprivileged userspace from areas such as Javascript executing in web-browser. Moreover, the study present sophisticated security implications of non-moveable interrupts such as, soft-irqs and rescheduling interrupts. One way to countermeasure non-movable interrupts based attack vectors is to generate arbitrary interrupts at random times to maintain ample noise floor within a system to hamper any attack attempts. To this end, the study further presents a framework to analyze, address and strengthen the security posture in aforementioned context.

Liu et al. [S27] address the vulnerability regarding the instruction cache (I-cache), which leaks secret information in cloud-based environments across different virtual machines. In such exploitations, the sensitive information is leaked through I-cache due to the secret-dependent execution paths the cipher takes. The authors argue that these attacks pose a serious threat and evaluate the degree of suitability of various randomized-mapping schemes as countermeasures. Given a set of randomized mapping schemes, the proposed approach leverages machine learning to build a classification matrix to quantitatively characterize the strength of a given scheme against an I-cache attack. Although, there is no silver bullet to the mitigation, the proposed approach

enables the practitioners to craft an adaptive solution for an effective defense while taking into consideration the trade-offs between the level of security and the runtime overheads.

Hsaio et al. [S52] also address the vulnerability of the virtual machines to cross-VM cache attacks in the cloud environments. The authors propose an approach to instrument the hypervisors utilizing the Intel VT-x extension, to monitor the memory allocation and access behavior of the guest VMs at runtime. In particular, a novel hardware-assisted MMU (memory management unit) redirection mechanism is presented, which allows the monitoring and profiling of all the memory accesses made on behalf of the guest VMs within the hypervisor memory space by transparently intercepting the accesses. Note that although the proposed approach needs to modify neither the guest nor the host VM, the instrumentation of the hypervisor is required. Furthermore, depending upon the configurable trade-off between the level of security and performance, some use-case sensitive defense postures can be delivered.

Deutsch et al. [S90] present a performant mitigation approach to memory based timing channels where an attacker utilizes contention to exfiltrate sensitive information. The study elaborates that existing countermeasure approaches can occasionally be effective however, they suffer from severe performance penalties. Among such approaches bandwidth partitioning and intensive profiling are notables ones however, they remain prone to fine-grained timing exploitations. To this end, the study presents an approach to deliver good performance, dynamic reshaping of memory traffic and protects against timing channels. At its heart, the proposed approach utilizes a novel directed acyclic graph based traffic representation and locates the paths which experience significant contention and employ heuristic based dynamic partitions to reshape, group and adaptively delay culprit traffic patterns while, ensuring absence of timing channels. Furthermore, the study formally verifies the security of aforementioned approach and provides strong security guarantees. Last but not least, presented approach is claimed to be generic enough so it can be extended to defend against scheduler-based timing channels.

Belleville et al. [S24] present a preventive countermeasure to harden the programs against the microarchitectural side-channel attacks. The proposed approach utilizes code polymorphism as a mechanism to add unpredictability in sensitive code segments. Furthermore, lightweight runtime code generation, on top of static code optimization during compilation, is used to further strengthen the program executions. The empirical results confirm that the programs treated with the aforementioned approach exhibit strong security characteristics while having acceptable runtime overheads.

Brosch et al. [S85] put forward the case of protecting privacy of trained neural networks (NN) which recent research has revealed to be prone to a wide range of microarchitecture side-channel attacks, through which attacker aims to infer latent parameters. Embedded/IoT devices typically are employing these trained NN models in specific application scenarios but, due their constrained designs become a prime victims of aforementioned attacks. Typically, a NN evaluation for given input comprises sequential execution of neurons within a hidden layer where, each neuron performs one or more sequential multiplications which give rise to deterministic patterns from an attacker's point of view. To counteract, the study proposes that execution order of neurons within same layer should be altered each time and the multiplication order within each neuron to be randomized. Overall, the cumulative effect poses a barrier for an attacker, to identify at which point in time a neuron is executed and when which input was multiplied with corresponding weight. The experimental results demonstrate that proposed technique significantly hampered attack's success in a number of scenarios.

2) PREDICTION/PREVENTION OF ZERO-DAY ATTACKS

Yuce et al. [S6] present a number of countermeasures against the zero-day fault injection-based attacks on the embedded/IoT processors. Fault injection remains an effective vector of attack, which operates by perturbing the ongoing computations, such as cryptographic operations. These perturbations, in effect, derail the sanity of the processor's internal state, which opens up potential side-channels for adversarial exploitation. The proposed approach leverages three techniques: fault detection, critical state checkpointing, and rolling back to last known good state of the computation in the event of a fault. The authors argue that adopting the aforementioned techniques in a microarchitectural design would ensure effective resilience on the hardware side against a range of fault injection-based attacks, including clock glitches, voltage starvation, and EM pulses. Similarly, on the software side, a set of custom instructions can strategically be placed inside the sensitive code regions to minimize the pollution in the internal state of the microarchitectural elements in lieu of certain events, such as exceptions. The aforementioned countermeasures can help avoid potentially exploitable states, through which the attacks can be carried out.

Guo et al. [S15] present an approach to detect the program paths that can potentially lead up to the timing-based exploitation of the CPU caches. The authors point out that the standard static analysis approaches may be used to determine whether a program is secure against the aforementioned exploitations. However, these guarantees remain short when it comes to the speculative execution scenarios. To this end, the proposed approach uses symbolic execution for state-space exploration tailored to speculative program execution, which aims to discover the paths that could potentially be exploited. Avoiding these paths during the speculative or normal executions would then guarantee the safety of the system against the aforementioned timing attacks, including the zero-day attacks.

Lang et al. [S72] present an approach to safeguard the Intel SGX enclaves against all the known access- and trace-driven cache attacks. The proposed approach aims to monopolize the whole CPU during the security-critical computations inside the enclaves by spawning a number of dummy SGX

TABLE 4. Primary studies which employ machine learning in (O)ffense and (C)ountermeasure roles, accompanying a short summary and their publication year.

Primary Study	Year	Role	Context of Usage
[S27]	2015	C	ML assists to rank resilience of a number of cache randomization schemes against a set of attacks.
[S53]	2021	O	Exfiltrating latent parameter of trained ML model from congestion in PCIe traffic
[S93]	2022	O	Leverage a trained ML model to disassemble instructions under execution from side-channel leakage
[S91]	2022	O	user keystrokes inferral from observable change in CPU operating frequency
[S89]	2022	O	To better understand attack-mechanics from a large pool of behavior observations of a system and to leverage this knowledge to further conceal the attack

threads. These threads are then closely monitored against the violations of exclusive scheduling, abrupt termination, and exit events. The attacks, including the zero-day attacks, are prevented by starving the adversaries during the secure computations.

Chen et al. [S47] also present an approach to safeguard the Intel SGX enclaves against the speculative execution attacks, including Foreshadow [36]. The proposed approach augments the sensitive code regions to protect the computations carried out in the enclaves. The external interrupts are also temporarily disabled during the computations inside the enclaves, which are, indeed, one of the prime enablers for the attacks of interest. Since the proposed approach establishes a secure environment even during speculative execution, it curbs the possibility of successfully carrying out the attacks, including zero-day attacks.

3) FURTHER DISCUSSION

Cache-focused countermeasures remain a prominent and recurrent theme as cache-based side-channels are frequently exploited in the attack dynamics regardless of the underlying hardware type, such as general-purpose-computing-based, cloud-computing-based, or embedded/IoT hardware. Moreover, cache exploitation, even in the speculative execution flows, is more troublesome as the potential countermeasures typically penalize the system's performance.

An emerging theme among microarchitecture attacks and defences is employment and utilization of machine learning as an artificial intelligence approach in recent years. We observe that overall 4% (5 out of 104) of primary studies belong to this theme spanning across years 2015-2022 and we anticipate this trend to accelerate. Factors like generative AI [37] will certainly be among the forefront of discovering new microarchitecture vulnerabilities and perhaps generate after-fixes on-the-fly with automated reasoning. With sustained miniaturization in chip fabrication and advances in lithographic techniques, we anticipate newer generation of chips with affordable onboard AI hardware being readily available, which could be programmed for any application-specific purpose including security. However, it could also be a double-edged sword where in the wrong hands could lead upto new generation of smart and sophisticated attacks that were never seen before. However, to date Table 4 provides the subset of primary studies which leveraged machine learning both for the offense and countermeasures. Notably, we observe that research inclination is somewhat more towards offensive approaches as, 4 out of 5 studies fall in this category. A frequently reported drawback is high computational cost of employing ML, which although could be tolerable for workstation grade systems however, usually not for embedded/IoT devices.

Another recurring theme is securing the trusted computing mechanisms, such as Intel SGX. Despite the vendor's tight security assurances, researchers have been able to circumvent and compromise these assurances. In this regard, several mitigation strategies have been proposed. These strategies, however, incur performance overheads. Therefore, until the CPU vendors patch/update their microarchitectures, the trusted computing mechanisms will deliver little faith in their security guarantees to the end users.

D. RQ4: GIVEN THE PUBLISHED COUNTERMEASURES, HOW SECURE A SYSTEM WE CAN BUILD AGAINST MICROARCHITECTURE SIDE-CHANNEL ATTACKS AND WHAT LESSONS WE CAN INCORPORATE IN THIS SYSTEM-DESIGN PROCESS?

To guide the security-oriented architectural process, we have identified the relevant primary studies and grouped them in the following categories: *Metrics, Modeling, and Assessment* – grouping the studies that guide the architectural process to appropriately model and quantitatively assess the security postures of the microarchitectural components; *Design flow and Synthesis* – grouping the studies that aid hardware designers to integrate security evaluation techniques to assess the security postures during the design phase; *Verification* – grouping the studies that help the designers verify the intended security postures of the designed components; and *Miscellaneous* – grouping the studies that do not belong to any of the categories above, but are relevant in the scope of the current research question.

1) METRICS, MODELING, AND ASSESSMENT

He et al. [S30] focus on the security assessment of caches. Existing research proposes a number of secure cache architectures to address the cache-related security flaws. The authors, however, argue that reliable methods for assessing the strengths of cache architectures remain unavailable. To this end, they propose a probabilistic information flow graph (PIFG) to model the interactions between a given cache architecture, a malicious program, and a victim program. At a very high level, the PIFG model is used to quantify the resilience of a cache architecture against the attacks. The

effectiveness of the proposed metric has been thoroughly evaluated on nine different cache architectures. Note that the lack of such a metric would left the researchers to rely on simulations or hardware instrumentation-based approaches to assess the cache security.

Deng et al. [S26] propose an approach to determine and thoroughly explore whether a given cache architecture is prone to the timing-channel vulnerabilities. Their approach is based on bounded model checking utilizing the computation tree logic to model the execution paths regarding the interactions between the processors and the caches, such that logic formulas can be derived for the vulnerable paths leading up to the timing side-channels. Overall, the proposed approach uncovered 28 different types of existing attacks, including 8 novel variations.

Callan et al. [S5] propose a metric, called *SAVAT*, to quantitatively measure the side-channel signal by quantifying the single-instruction difference among the executions of otherwise two identical programs. This metric is significant in the sense of its granular applicability to the level of difference of a single instruction execution, which can account for a large variety of microarchitectural and electronic activity. Therefore, the proposed metric can be utilized to pinpoint the vulnerable aspects of a microarchitecture under the influence of program execution, which would help designers make informed decisions when addressing the vulnerable aspects of an existing design.

Yilmaz et al. [S42] develop a metric to quantitatively measure the amount of information to be transmitted by the execution of a particular sequence of instructions on a CPU. The proposed metric is significant in regards to aiding the software and hardware designs to minimize the inadvertent side-channel leakage. Furthermore, leveraging this metric would also help identify the vulnerable portions of the programs involved in the leaks.

2) DESIGN FLOW AND SYNTHESIS

Barenghi et al. [S76] present an approach for strengthening the existing FPGA design flows with integrated side-channel security mechanisms. FPGAs, being programmable hardware, are frequently used to synthesize soft-core processors. The study argues that having earlier security-oriented feedback during the design phase would help the designers develop security-hardened designs. The proposed approach detects the side-channel leakage during the post-synthesis and the post-implementation phases of the hardware design. Furthermore, it provides precise insights into the sources of information leakage at the level of microarchitectural components. The evaluations of a cryptographic hardware uncovered several design vulnerabilities and aided in rectifying the design.

Arsath et al. [S54] propose to have a module-wise microarchitectural security audit for the processor designs. The notion behind the proposed approach is to ultimately establish a methodology that universally safeguards all the sensitive applications to be executed on a given processor. Moreover, such a ground-up design approach should lead to minimal performance degradation and reduce the power as well as the area overheads to the extent possible. As a use case, the study focuses on the resilience of an open source RISC CPU against the power-consumption side-channel attacks. The CPU is first analyzed by running a set of benchmark applications and its overall power consumption profile is correlated to the degree of information leakage. Then, given the collected data and the source RTL (Register-transfer Level) of the design, the proposed approach analyzes the leakage on a per hardware module basis, such as the register banks and the pipeline buffers. The empirical evaluations demonstrate that the predicted degree of leakage of a module is generally in direct proportion to the module's vulnerability for exploitation. Such leakages often originate as a byproduct of the automated translations in EDA (electronic design automation). Although such translations are functionally correct, they are often prone to the security flaws, which need to be mitigated through a redesign. The aforementioned study also advocates for the data path obfuscation as it significantly helps reduce information leakage.

Hur et al. [S96] emphasize that critical microarchitecture vulnerabilities need to be discovered and addressed during the RTL development stage of a CPU design. In this regard, transient execution vulnerabilities are the most serious which could have critical security impacts on software systems. To this end, this study leverages fuzz testing [38] to aid discovering and subsequent remedial of aforementioned vulnerabilities in a two step process. In the first step, templates of known vulnerabilities (such as Meltdown [1]) are utilized and all potential variations of a particular attack dynamics are exercised, to establish whether the design in question is prone specifically to this attack. In the second step, different combinations of known vulnerabilities are specified as design approval constraints, to establish whether given RTL design is vulnerable to a particular combination and if so, the approach outputs the test-case exercising the culprit vulnerability, which aids the designer to rectify the affected areas of his design. Furthermore, the presented approach has showcased its strengths by discovering two novel vulnerabilities affecting a RISC-V microarchitecture. The study also emphasizes on leveraging fuzz testing as an indispensable tool to test security posture of any non-trivial RTL design.

He et al. [S95] argue that cryptographic hardware design revisions at register-transfer-level (RTL) often do not take into account fine-grained security enhancements and thus vulnerable components remain exploitable in the future. To address, the study proposes a security-aware hardware-design, synthesis and optimization framework with evaluation metrics. In a nutshell, implementation details of hardware RTL design are analyzed at first, followed by observations made in an environment while, being subjected to leakage-causing inputs. The degree of observed leakage is quantitatively assessed and vulnerable sub-modules in the design are identified. The study proposes to employ conditional-hiding in contrast to masking schemes, as an effective approach to curb leakage. Conditional hiding refers

to re-architecting those registers (which store or process sensitive information), such that their behavioral patterns remain consistent regardless, to change in input patterns. This is usually realized by governing multiple registers, under a load-balancing strategy. Lastly, the proposed framework demonstrates its strengths through simulations and FPGA based hardware synthesis, for a number of AES cryptographic test scenarios.

Lee et al. [S49] exclusively highlight the relevance of ECC (elliptic curve cryptography) on portable and IoT devices. However, ECC is generally computationally intensive. Therefore, some dedicated hardware is typically needed for sufficient performance. Furthermore, when ECC is implemented on IoT and embedded hardware, a mathematically limited flavor of ECC, such as a single finite field, is often employed. The authors present a hardware-efficient and secure design to support a robust ECC implementation, which is also resilient against the attacks. The proposed solution relies on a single-chip, heterogeneous dual-processing-element (dual-PE) architecture capable of providing various types of PEs leveraging the full pipelining. The authors also suggest that the aforementioned design can further benefit from a two-level memory hierarchy with a local memory synchronization scheme.

Bache et al. [S60] propose a methodology towards having a timing and power side-channel resistant cryptographic processor based on ARX-cryptography, which utilizes only the addition, rotation, and xor (ARX) operations. The adequate utilization of the ARX-based computations ensures ample confusion and diffusion properties. Although naive ARX-based implementations are prone to the power attacks, they are robust against other classes of attacks, such as the timing attacks. They are, therefore, often deemed desirable in IoT and smart card applications. The authors address the aforementioned limitations and propose a novel application-specific processor design based on the ARX theory. At a very high level, the proposed approach provides security against the timing and power attacks by protecting the data paths with a custom boolean masking scheme. Furthermore, the processor-specific instruction set used in the proposed masking scheme, follows the principles of Threshold Implementation [39] for provable security. Last but not least, the approach incurs moderate costs compared to more sophisticated and protected hardware implementations.

Borrello et al. [S104] advocate for constant-time implementation of cryptographic implementations as effective deterrence against speculative execution attacks. However, they argue that developing quality constant-time code by hand is difficult and implementations from real world revealed many flaws. Moreover, automated tools to transform a given implementation into constant-time implementation offered, limited security in production grade software. To this end, the study introduces a compiler based automated tool to harden programs against a range of microarchitecture attacks. At its core, the proposed tool strives to completely linearize secret-dependent control- and data-flows. However, doing so in real world can result in state explosion. To address this challenge, optimizations such as just-in-time loop linearization and excessive function cloning is used. The resultant implementation although provides hardened security on real world software albeit, with moderate performance overhead.

Kiaei et al. [S61] present a software synthesis technique for time-sensitive embedded applications. The study argues that in certain application scenarios, the strict execution-time requirements are not mandatory as long as the imposed real-time deadlines are met. In cryptographic applications, to prevent the timing side-channels, the implementation demands to be data-independent and should follow the precise execution timings. To this end, the study proposes a parallel synchronous programming model, which delivers high throughput under modest time constraints for the aforementioned application scenario. The approach automatically transforms a given program for a predetermined number of parallel threads of execution, which coordinate synchronously. This coordination ensures the elimination of the contention while providing high throughput and meeting the timing deadlines. Furthermore, the study illustrates that the proposed execution model can also curb the timing side-channels.

Oh et al. [S37] discuss a hardware-based defense mechanism for mitigating the access-based side-channel attacks under speculative execution in the Intel SGX enclaves. The authors discuss that the state-of-the-art mitigation is based on the ORAM (Oblivious RAM) protocol, which, although, adequate, incurs significant overhead. The proposed approach is based on the PCI plug-in hardware implemented in FPGA, providing a trusted storage service in a completely isolated and secure environment. The service securely keeps the SGX authentication secrets and verifies the authenticity as well as the authorization for the communication between the program running in an enclave and the storage device implemented in FPGA. The empirical results suggest that the FPGA-based solution outperforms the three state-of-the-art ORAM-based approaches. The proposed approach could also serve as a beacon to adopt a synthesized hardware-based security solution where software-based solutions are simply too prohibitive in terms of overall performance.

Barenghi et al [S59] argue that changes in the CPU microarchitecture and ISA can manifest into side-channel leakage. Even innocuous changes, such as modifying the register allocation order, can lead to an exploitable vulnerability. Therefore, the microarchitectural features of the target CPU should be kept in consideration while assessing the side-channel leakage behavior of a software implementation. Moreover, the authors discover that the contention inside the pipeline buffers can cause critical information leakage, which could never have been spotted through mere static analysis.

Vougiokas et al. [S41] provide design lessons on performant, attack-resistant branch predictors. The authors argue that the existing approaches sacrifice performance gains achieved by branch prediction for side-channel security. For example, to remain secure, the existing approaches often liberally rely on flushing the internal prediction states.

To this end, the authors introduce a quantitative metric, called *transient-state prediction accuracy* (TIPA), aiming to rank the side-channel free branch predictor designs for the future. The motivation behind TIPA lies on the idea that flushing the branch predictor during the transient states, affects the performance far worse than flushing it during the steady states. Leveraging TIPA would aid in designing safer branch predictors of the future.

3) VERIFICATION

Eldib et al. [S29] describe an approach to formally assess the degree of security provided by a countermeasure technique. The study focuses on data-masking techniques as an effective countermeasure approach to the power-based side-channel attacks against the cryptographic applications. Data masking is often relied upon to decouple the statistical dependence of sensitive data to its corresponding side-channel emissions, which is an important mitigative step. However, the study argues that the sensitive data is desired to be perfectly masked. The selection of a suitable masking technique for a particular use case, on the other hand, remains to be a research problem. To this end, the study proposes an SMT-based (i.e., a satisfiability modulo theory-based) method to formally verify the degree of security provided by a given masking technique, where the verification problem is translated into a sequence of satisfiability problems. The underlying notion is to check whether any intermediate computation statistically depends upon the contents of the sensitive data.

Grandmaison et al. [S94] present a framework to address security reduction in software masked implementations which, originates due to mismatch between leakage sources considered in the security proof and actual sources of leakage in microarchitecture. The presented approach addresses this limitation by taking into account detailed models of microarchitectural components and their leakage profiles and synthesizes automated test cases to uncover leakage scenarios. Furthermore, discussed approach formally verifies a given software implementation through, binary analysis and identifies, the chain of events and sequence of instructions which would lead upto opening of windows for information leakage. The experiments conducted on an ARM CPU validate strengths of their presented approach.

Colvin et al. [S17] argue that speculative execution makes it difficult to formally reason about the security properties of a software system. Such a void has, therefore, left the software systems to fall victim to a wide spectrum of attacks, including Spectre and Meltdown, which are attributed to speculative execution. Similarly, the role of cache in all these attacks is even a more difficult problem to be formally reasoned about. To address these issues, the authors present high-level abstract semantics to formalize speculative execution and its side effects. The proposed semantics is found to be effective in discovering the sources of information leaks caused by speculative execution. For example, the empirical studies suggest that had the proposed approach been used, Spectre could have been discovered early on during a security assessment. Another study [S22] deals with the same problem by following a similar approach. In particular, a realistic model of speculative execution is transformed into an abstract one, which is then simplified and refined under formal verification by using a standard model checker. The practicability of the proposed approach is demonstrated by discovering some security leaks in a pipelined RISC processor.

4) MISCELLANEOUS

Regazzoni et al. [S4] focus on the emerging paradigm of *approximate computing*, which is an architectural paradigm where limited and controlled errors are tolerated during computations. Although more research in this field is needed, the proposed paradigm can help build faster, smaller, more cost-effective, and less power-consuming hardware circuits. From the perspective of security, approximate computing can deliver resistance to the fault injection as well as the side-channel attacks.

Grimsdal et al. [S69] explore the feasibility of adopting micro-kernels to thwart microarchitectural attacks. With the help of the empirical studies where three micro-kernels with strict process isolation mechanisms have been evaluated, the authors conjuncture that strict process isolation is effective in limiting the effectiveness of the microarchitectural attacks, such as Meltdown and Spectre.

Nabeel et al. [S11] advocate for the need to have new processor designs that natively provide support for data privacy through cryptography. They argue that design optimizations in recent years paved the ways for unique microarchitectural vulnerabilities, which caused attacks, such as Meltdown and Spectre. They present their findings with regard to a secure co-processor design aimed at delivering privacy through leveraging homomorphic encryption in the data paths. In the empirical studies, they opt to integrate their design as a co-processor linked through the communication bus to the main processor. Depending on the data privacy settings, the main processor can opt to delegate the sensitive computations to this co-processor to guarantee data security.

Mao et al. [S57] advocate for onboard reconfigurable hardware architectures capable of on-demand software configuration. Software-based mitigations may have an impact on the system's performance. A hardware-based solution, on the other hand, would be flexible and circumvent the runtime overheads. The authors first synthesize a soft-core processor on an FPGA board, whose architecture and features can be reconfigured through software. The processor is then configured to create a hardware attack detection module, which also delivers tailored hardware-based mitigation for cache-based timing attacks.

Seuscheck et al. [S14] highlight that embedded/IoT systems remain vulnerable to side-channel attacks. Although masking scheme-based mitigations are generally effective, they remain prone to unwanted correlation-based leakage through different registers on the same shared CPU. The authors do not specify a particular type of leakage, but rather describe the leakage at an abstract level to cater for any side-channel leakage that can be correlated with

TABLE 5. Table of articles which are selected as primary studies.

Study ID	Title
S1	A Comprehensive Side-Channel Information Leakage Analysis of an In-Order RISC CPU Microarchitecture [40]
S2	An Architecture-Independent Instruction Shuffler to Protect against Side-Channel Attacks [41]
S3	Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs [42]
S4	Side Channel Attacks vs Approximate Computing [43]
S5	A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events [44]
S6	A Secure Exception Mode for Fault-Attack-Resistant Processing [45]
S7	Authentication Control Point and Its Implications For Secure Processor Design [46]
S8	BranchSpec: Information Leakage Attacks Exploiting Speculative Branch Instruction Executions [7]
S9	CheckMate: Automated Synthesis of Hardware Exploits and Security Litmus Tests [47]
S10	Composable Template Attacks Using Templates for Individual Architectural Components [48]
S11	CoPHEE: Co-processor for Partially Homomorphic Encrypted Execution [49]
S12	Fooling the Sense of Cross-Core Last-Level Cache Eviction Based Attacker by Prefetching Common Sense [50]
S13	Leaky Buddies: Cross-Component Covert Channels on Integrated CPU-GPU Systems [51]
S14	Side-Channel Leakage Models for RISC Instruction Set Architectures from Empirical Data [52]
S15	SPECUSYM: Speculative Symbolic Execution for Cache Timing Leak Detection [53]
S16	XDIVINSA: eXtended DIVersifying INStruction Agent to Mitigate Power Side-Channel Leakage [54]
S17	An Abstract Semantics of Speculative Execution for Reasoning About Security Vulnerabilities [55]
S18	C5: Cross-Cores Cache Covert Channel [10]
S19	CCCiCC: A Cross-Core Cache-Independent Covert Channel on AMD Family 15h CPUs [56]
S20	Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript [57]
S21	Leaky Controller: Cross-VM Memory Controller Covert Channel on Multi-core Systems [58]
S22	Model Checking Speculation-Dependent Security Properties: Abstracting and Reducing Processor Models for Sound and Complete Verification [59]
S23	A Novel Covert Channel Attack Using Memory Encryption Engine Cache [60]
S24	Automated Software Protection for the Masses Against Side-Channel Attacks [61]
S25	BranchScope: A New Side-Channel Attack on Directional Branch Predictor [62]
S26	Cache Timing Side-Channel Vulnerability Checking with Computation Tree Logic [63]
S27	Can Randomized Mapping Secure Instruction Caches from Side-Channel Attacks? [64]
S28	Cracking Randomized Coalescing Techniques with An Efficient Profiling-Based Side-Channel Attack to GPU [65]
S29	Formal Verification of Software Countermeasures against Side-Channel Attacks [66]
S30	How Secure is Your Cache against Side-Channel Attacks? [67]
S31	Klotski: Efficient Obfuscated Execution against Controlled-Channel Attacks [68]
S32	MicroScope: Enabling Microarchitectural Replay Attacks [11]
S33	On the Detection of Applications in Co-Resident Virtual Machines via a Memory Deduplication Side-Channel [69]
S34	Power Side Channel Attack Analysis and Detection [70]
S35	Rendered Insecure: GPU Side Channel Attacks Are Practical [71]
S36	Streamline: A Fast Flushless Cache Covert-Channel Attack by Enabling Asynchronous Collusion [72]
S37	TRUSTORE: Side-Channel Resistant Storage for SGX Using Intel Hybrid CPU-FPGA [73]
S38	A Fetching Tale: Covert Communication with the Hardware Prefetcher [74]
S39	A New Side-Channel Vulnerability on Modern Computers by Exploiting Electromagnetic Emanations from the Power Management Unit [75]
S40	A Time-/Frequency-Domain Side-Channel Attack Resistant AES-128 and RSA-4K Crypto-Processor in 14-nm CMOS [76]
S41	BRB: Mitigating Branch Predictor Side-Channels. [77]
S42	Capacity of the EM Covert/Side-Channel Created by the Execution of Instructions in a Processor [78]

TABLE 5. *(Continued.)* **Table of articles which are selected as primary studies.**

S43	CIDPro: Custom Instructions for Dynamic Program Diversification [79]
S44	Cross-VM Cache Attacks on AES [80]
S45	DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors [81]
S46	Defeating Hardware Prefetchers in Flush+Reload Side-Channel Attack [82]
S47	Defeating Speculative-Execution Attacks on SGX with HyperRace [83]
S48	Efficient and side-channel-secure block cipher implementation with custom instructions on FPGA [84]
S49	Efficient Power-Analysis-Resistant Dual-Field Elliptic Curve Cryptographic Processor Using Heterogeneous Dual-Processing-Element Architecture [85]
S50	EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals [86]
S51	Flush-Detector: More Secure API Resistant to Flush-Based Spectre Attacks on ARM Cortex-A9 [87]
S52	Hardware-Assisted MMU Redirection for In-Guest Monitoring and API Profiling [88]
S53	Invisible Probe: Timing Attacks with PCIe Congestion Side-channel [89]
S54	PARAM: A Microprocessor Hardened for Power Side-Channel Attack Resistance [90]
S55	Partial-SMT: Core-Scheduling Protection Against SMT Contention-Based Attacks [91]
S56	Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors [92]
S57	REHAD: Using Low-Frequency Reconfigurable Hardware for Cache Side-Channel Attacks Detection [93]
S58	Reinforcing Meltdown Attack by Using a Return Stack Buffer [94]
S59	Side-channel security of superscalar CPUs : Evaluating the Impact of Micro-architectural Features [95]
S60	SPARX — A side-channel protected processor for ARX-based cryptography [96]
S61	Synthesis of Parallel Synchronous Software [97]
S62	Towards a New Thermal Monitoring Based Framework for Embedded CPS Device Security [98]
S63	A Bit-Level Approach to Side Channel Based Disassembling [99]
S64	A Study on Microarchitectural Covert Channel Vulnerabilities in Infrastructure-as-a-Service [100]
S65	A Vulnerability in RSA Implementations Due to Instruction Cache Analysis and Its Demonstration on OpenSSL [101]
S66	An Efficient Approach for Mitigating Covert Storage Channel Attacks in Virtual Machines by the Anti-Detection Criterion [102]
S67	Cache-Zoomer: On-demand High-resolution Cache Monitoring for Security [103]
S68	Calibration Done Right: Noiseless Flush+Flush Attacks [104]
S69	Can Microkernels Mitigate Microarchitectural Attacks? [105]
S70	CHASM: Security Evaluation of Cache Mapping Schemes [106]
S71	Differential Analysis and Fingerprinting of ZombieLoads on Block Ciphers [107]
S72	E-SGX: Effective Cache Side-Channel Protection for Intel SGX on Untrusted OS [108]
S73	Efficient Information-Flow Verification Under Speculative Execution [109]
S74	Flush+Flush: A Fast and Stealthy Cache Attack [110]
S75	HexPADS: A Platform to Detect “Stealth” Attacks [111]
S76	Integrating Side Channel Security in the FPGA Hardware Design Flow [112]
S77	Malware Guard Extension: abusing Intel SGX to conceal cache attacks [113]
S78	MemJam: A False Dependency Attack Against Constant-Time Crypto Implementations [114]
S79	NetSpectre: Read Arbitrary Memory over Network [115]
S80	New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures [116]
S81	New Results on Instruction Cache Attacks [117]
S82	Off-Limits: Abusing Legacy x86 Memory Segmentation to Spy on Enclaved Execution [118]
S83	Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications [119]
S84	Validation of Abstract Side-Channel Models for Computer Architectures [120]
S85	Counteract side-channel analysis of neural networks by shuffling [121]
S86	SpecSafe: detecting cache side channels in a speculative world [122]
S87	Validation of Side-Channel Models via Observation Refinement [123]

TABLE 5. *(Continued.)* **Table of articles which are selected as primary studies.**

S88	Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy [124]
S89	There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack [125]
S90	DAGguise: mitigating memory timing side channels [126]
S91	DF-SCA: Dynamic Frequency Side Channel Attacks are Practical [127]
S92	CINI MINIS: Domain Isolation for Fault and Combined Security [128]
S93	A deep-learning approach to side-channel based CPU disassembly at design time [129]
S94	ARMISTICE: Microarchitectural Leakage Modeling for Masked Software Formal Verification [130]
S95	Security Oriented Design Framework for EM Side-Channel Protection in RTL Implementations [131]
S96	SpecDoctor: Differential Fuzz Testing to Find Transient Execution Vulnerabilities [132]
S97	Microarchitectural Leakage Templates and Their Application to Cache-Based Side Channels [133]
S98	PREFENDER: A Prefetching Defender against Cache Side Channel Attacks as A Pretender [134]
S99	Frequency Throttling Side-Channel Attack [135]
S100	PACMAN: attacking ARM pointer authentication with speculative execution [136]
S101	Delay-on-Squash: Stopping Microarchitectural Replay Attacks in Their Tracks [137]
S102	Write Me and I'll Tell You Secrets – Write-After-Write Effects On Intel CPUs [138]
S103	SeBROP: blind ROP attacks without returns [139]
S104	Constantine: Automatic Side-Channel Resistance Using Efficient Control and Data Flow Linearization [140]

the contents of the registers. Software-based cryptographic implementations on embedded/IoT systems remain particular candidates for exploitations through this type of leakage. The study empirically demonstrates that an arbitrary mapping of the registers to the intermediate values of a software cipher can cause information leaks. As a mitigation technique, a compile-time tool is presented, which sensibly maps the intermediate values to the registers with the goal of minimizing the aforementioned correlation.

Feldtkeller et al. [S92] bring forward an important threat vector to cryptographic implementations with regards to combination of side-channel vulnerability discovery, analysis and tailored fault-injection attack strategies. The study explains that on individual basis aforementioned vectors are well-researched however, attempts to combine them is mostly unexplored, yet quite potent when employed. To this end, the study presents a mitigative formal framework, built upon the ideas of domain isolation and secure composition to construct, an essential set of hardware-level building blocks which can provide resilience against faults and when employed in a collaborative-chain, can effectively hamper an attacker using combination of tactics. Lastly, the study outlines that although proposed approach has significant overhead costs but nonetheless to-date an only effective remedy against outlined attacks.

5) OUR THOUGHTS

Architecting systems that are prone to microarchitectural exploitations in both arenas of hardware and software remains an actively researched topic. However, as more and more exploitations are being discovered and remedied, they are also shaping the security-oriented design practices, which are improving over time. Furthermore, establishing a fine trade-off between system performance and security still remains to be a difficult problem as these facets are often orthogonal. Designers are often faced with the challenges to produce a design under strict time and budget constraints. However, as further research continues, lessons are being learned and quantitative metrics to evaluate the security-oriented aspects of the designs are being proposed and utilized. We are hoping in the future that more refined and broad-spectrum security-oriented design processes become the common norm.

VII. THREATS TO VALIDITY AND FUTURE WORK

Systematic mapping studies capture the research focus and the trends within the literature. They, however, do not delve into the details of the findings reported by the primary studies. Although, we, as researchers, tend to implicitly pick the quality works as the primary studies to be analyzed, we are often limited by the selection process and the classification criteria. Therefore, if desired, a focused literature review with a narrower scope is recommended.

Moreover, systematic mapping studies are empirical in nature. Therefore, they typically suffer from some threats to validity. To address the threats to validity surrounding the selection, screening, and the classification processes, we opted to follow the footsteps of the previously published mapping studies. Yet, we chose to flex and/or adopt the footsteps, tailored to suit the focus and the scope of our work. We also relied on more than one iteration during the selection, screening, and the classification process to minimize the potential misjudgments. Furthermore, we also utilized team discussions as needed to reach on mutual consensus. We used three well-known and frequently-used scientific databases. We, however, do not neglect the possibility that had we relied on more than three research databases could have further improved the quality of our discussions or expanded the selection of primary studies.

TABLE 6. Abbreviations and their expansion which are used in this article.

Abbreviation	Expansion
AES	Advance Encryption Standard
ALU	Arithmetic And Logic Unit
API	Application Programming Interface
ARX	Addition Rotation And Xor (exclusive Or)
BITP	Back-invalidation Hits Triggered Prefetching
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DES	Data Encryption Standard
DPL	Dual-rail Precharge Logic
ECC	Elliptic Curve Cryptography
EDA	Electronic Design And Automation
EX/WB	Execution And Writeback
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
I-cache	Instruction Cache
IRQ	Interrupt Request
ISA	Instruction Set Architecture
IS/EX	Issue And Execution
LDO	Low-dropout Regulator
LFB	Line-fill Buffer
LLC	Last-level Cache
MEE	Memory Encryption Engine
ML	Machine Learning
MMU	Memory Management Unit
MOB	Memory Order Buffer
NN	Neural Networks
ORAM	Oblivious Ram
PCI	Peripheral Component Interconnect
PCIe	Pci Express
PE	Processing Element
PIFG	Probabilistic Information Flow Graph
RAM	Random Access Memory
ROP	Return Oriented Programming
RQ	Research Question
RSA	Rivest–Shamir–Adleman
RSB	Return Stack Buffer
RTL	Register-transfer Level
SDK	Software Development Kit
SGX	Software Guard Extensions
SMT	Simultaneous Multi Threading
TEE	Trusted Execution Environment
TIPA	Transient State Prediction Accuracy
VM	Virtual Machine
VRM	Voltage Regulator Module

Due to limited manpower and resources available at our expenditure, we did not had the opportunity to undertake multiple detailed reviews of complete set of papers. However, it is our firm belief that our selected pool of primary studies represents large and diverse aspects of microarchitecture security research, and does present an accurate and precise overall picture.

VIII. CONCLUDING REMARKS

This systematic mapping study presents a review of the state of knowledge from the curated set of primary studies in the field of microarchitecture side-channel security offenses, defenses, and measures to improve the security posture of computing systems. Directed by the review protocols, we relied on three standard databases for scientific literature

and located 546 articles, which further underwent screening within the protocol constraints identifying 104 articles as primary studies for this work. The classification scheme utilized seven classes for the task to locate the culprit components and vectors responsible for microarchitectural exploits. Furthermore, we also curated answers for our target research questions from the primary studies. We questioned and answered a number of aspects, including the aspects of microarchitectural leakage; how attack corridors have been created; the degree to which existing countermeasures are effective; and how to architect systems with improved resilience against these attacks. Moreover, we also identified the research spots where research emphasis had been present as well as the spots which may get further attention. Furthermore, we observed that over the course of the last five years there has been a linear growth in the number of publications, which directly reflects the importance and the attention of research being carried out in this domain.

We believe that our adopted classification scheme is suitable and fits the focus of this work well. This scheme is also highly reusable in the sense that it can be applied to capture new research trends by carrying out similar studies in the future. Last but not least, the inclusion of grey literature outside the academia, such as white papers, patents, webpages, and technical reports, could be valuable and reveal emerging themes rather quickly. However, such an effort is manually demanding, resource intensive, and perhaps would not be practical.

APPENDIX A
LIST OF PRIMARY STUDIES

See Table 5.

APPENDIX B
TABLE OF ABBREVIATIONS

See Table 6.

REFERENCES

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, Jun. 2018, pp. 973–990.

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1–19.

[3] B. Kitchenham, "Procedures for performing systematic reviews," Dept. Comput. Sci., Keele Univ., Keele, U.K., Tech. Rep. TR/SE-0401, 2004.

[4] A. K. Kanuparthi, R. Karri, G. Ormazabal, and S. K. Addepalli, "A survey of microarchitecture support for embedded processor security," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Aug. 2012, pp. 368–373.

[5] J. Szefer, "Survey of microarchitectural side and covert channels, attacks, and defenses," *J. Hardw. Syst. Secur.*, vol. 3, no. 3, pp. 219–234, Sep. 2019.

[6] 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.

[7] M. H. I. Chowdhuryy, H. Liu, and F. Yao, "BranchSpec: Information leakage attacks exploiting speculative branch instruction executions," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 529–536.

[8] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 603–615, Apr. 2015.

[9] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 679–697.

[10] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: Cross-cores cache covert channel," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 12th International Conference, DIMVA 2015, Milan, Italy, July 9–10, 2015, Proceedings 12*. Springer. 2015, pp. 46–64.

[11] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "MicroScope: Enabling microarchitectural replay attacks," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 318–331.

[12] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone explained: Architectural features and use cases," in *Proc. IEEE 2nd Int. Conf. Collaboration Internet Comput. (CIC)*, Nov. 2016, pp. 445–451.

[13] R. Wojtczuk and J. Rutkowska, "Attacking Intel trusted execution technology," *Black Hat DC*, vol. 2009, pp. 1–6, Feb. 2009.

[14] B. C. Xing, M. Shanahan, and R. Leslie-Hurd, "Intel software guard extensions (Intel SGX) software support for dynamic memory allocation inside an enclave," in *Proc. Hardw. Architectural Support Secur. Privacy*, Jun. 2016, pp. 1–9.

[15] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, Aug. 2015.

[16] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," School Comput. Sci. Math., Softw. Eng. Group, Keele Univ., Keele, U.K., EBSE Tech. Rep. EBSE-2007-01, 2007.

[17] A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 1–17, Jan. 2013.

[18] P. Meland, S. Tokas, G. Erdogan, K. Bernsmed, and A. Omerovic, "A systematic mapping study on cyber security indicator data," *Electronics*, vol. 10, no. 9, p. 1092, May 2021.

[19] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *J. Syst. Softw.*, vol. 117, pp. 334–356, Jul. 2016.

[20] *Google Scholar*. Accessed: Feb. 4, 2023. [Online]. Available: https://scholar.google.com

[21] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, May 2014, pp. 1–10.

[22] *Connected Papers|Find and Explore Academic Papers*. Accessed: Feb. 4, 2023. [Online]. Available: https://www.connectedpapers.com

[23] *IEEE Xplore Digital Library*. Accessed: Feb. 4, 2023. [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp

[24] *ACM Digital Library*. Accessed: Feb. 4, 2023. [Online]. Available: https://dl.acm.org/

[25] *Springer Link*. Accessed: Feb. 4, 2023. [Online]. Available: https://link.springer.com/

[26] Y. Zacchia Lun, A. D'Innocenzo, I. Malavolta, and M. D. Di Benedetto, "Cyber-physical systems security: A systematic mapping study," 2016, *arXiv:1605.09641*.

[27] P. H. Nguyen, S. Ali, and T. Yue, "Model-based security engineering for cyber-physical systems: A systematic mapping study," *Inf. Softw. Technol.*, vol. 83, pp. 116–135, Mar. 2017.

[28] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements Eng.*, vol. 11, no. 1, pp. 102–107, Mar. 2006.

[29] A. A. Ramaki, A. Rasoolzadegan, and A. G. Bafghi, "A systematic mapping study on intrusion alert analysis in intrusion detection systems," *ACM Comput. Surveys*, vol. 51, no. 3, pp. 1–41, May 2019.

[30] Y. Yarom, Q. Ge, F. Liu, R. B. Lee, and G. Heiser, "Mapping the intel last-level cache," *Cryptol. ePrint Arch.*, 2015.

[31] M. Prandini and M. Ramilli, "Return-oriented programming," *IEEE Secur. Privacy*, vol. 10, no. 6, pp. 84–87, Nov. 2012.

[32] 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*, May 2015, pp. 605–622.

[33] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A high resolution, low noise, L3 cache Side-Channel attack," in *Proc. 23rd USENIX Secur. Symp. (USENIX Secur.)*, 2014, pp. 719–732.

[34] L. Ren, C. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk, and S. Devadas, "Constants count: Practical improvements to oblivious RAM," in *Proc. 24th USENIX Secur. Symp. (USENIX Secur.)*, 2015, pp. 415–430.

[35] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom. (2021). *Prime+ Probe 1, JavaScript 0: Overcoming Browser-Based Side-Channel Defenses*. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/shusterman

[36] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 991–1008.

[37] I. Solaiman, "The gradient of generative AI release: Methods and considerations," 2023, *arXiv:2302.04844*.

[38] J. Liang, M. Wang, Y. Chen, Y. Jiang, and R. Zhang, "Fuzz testing in practice: Obstacles and solutions," in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2018, pp. 562–566.

[39] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup, "Threshold implementations of small S-boxes," *Cryptogr. Commun.*, vol. 7, no. 1, pp. 3–33, Mar. 2015.

[40] D. Zoni, A. Barenghi, G. Pelosi, and W. Fornaciari, "A comprehensive side-channel information leakage analysis of an in-order RISC CPU microarchitecture," *ACM Trans. Design Autom. Electron. Syst.*, vol. 23, no. 5, pp. 1–30, Sep. 2018.

[41] A. G. Bayrak, N. Velickovic, P. Ienne, and W. Burleson, "An architecture-independent instruction shuffler to protect against side-channel attacks," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 1–19, Jan. 2012.

[42] A. Fuchs and R. B. Lee, "Disruptive prefetching: Impact on side-channel attacks and cache designs," in *Proc. 8th ACM Int. Syst. Storage Conf.*, 2015, pp. 1–12.

[43] F. Regazzoni and I. Polian, "Side channel attacks vs approximate computing," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 321–326.

[44] R. Callan, A. Zajic, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 242–254.

[45] B. Yuce, C. Deshpande, M. Ghodrati, A. Bendre, L. Nazhandali, and P. Schaumont, "A secure exception mode for fault-attack-resistant processing," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 3, pp. 388–401, May 2019.

[46] W. Shi and H.-H. S. Lee, "Authentication control point and its implications for secure processor design," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 103–112.

[47] C. Trippel, D. Lustig, and M. Martonosi, "CheckMate: Automated synthesis of hardware exploits and security litmus tests," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 947–960.

[48] B. Liu, R. Lysecky, and J. M. Wang-Roveda, "Composable template attacks using templates for individual architectural components," in *Proc. IEEE 36th Int. Conf. Comput. Design (ICCD)*, Oct. 2018, pp. 1–8.

[49] M. Nabeel, M. Ashraf, E. Chielle, N. G. Tsoutsos, and M. Maniatakos, "CoPHEE: Co-processor for partially homomorphic encrypted execution," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Oct. 2019, pp. 131–140.

[50] B. Panda, "Fooling the sense of cross-core last-level cache eviction based attacker by prefetching common sense," in *Proc. 28th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2019, pp. 138–150.

[51] S. B. Dutta, H. Naghibijouybari, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Leaky buddies: Cross-component covert channels on integrated CPU-GPU systems," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 972–984.

[52] H. Seuschek and S. Rass, "Side-channel leakage models for RISC instruction set architectures from empirical data," in *Proc. Euromicro Conf. Digit. Syst. Design*, 2015, pp. 423–430.

[53] S. Guo, Y. Chen, P. Li, Y. Cheng, H. Wang, M. Wu, and Z. Zuo, "SpecuSym: Speculative symbolic execution for cache timing leak detection," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 1235–1247.

[54] T. H. Pham, B. Marshall, A. Fell, S.-K. Lam, and D. Page, "XDIVINSA: Extended diversifying instruction agent to mitigate power side-channel leakage," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2021, pp. 179–186.

[55] R. J. Colvin and K. Winter, "An abstract semantics of speculative execution for reasoning about security vulnerabilities," in *Proc. Formal Methods. FM Int. Workshops*, vol. 12233, 2020, pp. 323–341.

[56] C. D. Hailfinger, K. Lemke-Rust, and C. Paar, "CCCiCC: A cross-core cache-independent covert channel on AMD family 15h CPUs," in *Smart Card Research and Advanced Applications: 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11–13, 2019, Revised Selected Papers 18*. Springer, 2020, pp. 159–175.

[57] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript," in *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3–7, 2017, Revised Selected Papers 21*. Springer, 2017, pp. 247–267.

[58] B. Semal, K. Markantonakis, R. N. Akram, and J. Kalbantner, "Leaky controller: Cross-VM memory controller covert channel on multi-core systems," in *ICT Systems Security and Privacy Protection: 35th IFIP TC 11 International Conference, SEC 2020, Maribor, Slovenia, September 21–23, 2020, Proceedings 35*. Springer, 2020, pp. 3–16.

[59] G. Cabodi, P. Camurati, F. Finocchiaro, and D. Vendraminetto, "Model checking speculation-dependent security properties: Abstracting and reducing processor models for sound and complete verification," *Electronics*, vol. 11445, pp. 462–479, Sep. 2019.

[60] Y. Han and J. Kim, "A novel covert channel attack using memory encryption engine cache," in *Proc. 56th Annu. Design Automat. Conf.*, 2019, pp. 1–6.

[61] N. Belleville, D. Couroussé, K. Heydemann, and H.-P. Charles, "Automated software protection for the masses against side-channel attacks," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 4, pp. 1–27, Dec. 2018.

[62] D. Evtyushkin, R. Riley, N. C. A. E. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, Nov. 2018.

[63] S. Deng, W. Xiong, and J. Szefer, "Cache timing side-channel vulnerability checking with computation tree logic," in *Proc. 7th Int. Workshop Hardw. Architectural Support Security Privacy*, 2018, pp. 1–8.

[64] F. Liu, H. Wu, and R. B. Lee, "Can randomized mapping secure instruction caches from side-channel attacks?" in *Proc. 4th Workshop Hardware Architectural Support Security Privacy*, 2015, pp. 1–8.

[65] X. Wang and W. Zhang, "Cracking randomized coalescing techniques with an efficient profiling-based side-channel attack to GPU," in *Proc. 8th Int. Workshop Hardware Architectural Support Security Privacy*, 2019, pp. 1–8.

[66] H. Eldib, C. Wang, and P. Schaumont, "Formal verification of software countermeasures against side-channel attacks," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 2, pp. 1–24, Dec. 2014.

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

[68] P. Zhang, C. Song, H. Yin, D. Zou, E. Shi, and H. Jin, "Klotski: Efficient obfuscated execution against controlled-channel attacks," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 1263–1276.

[69] J. Lindemann and M. Fischer, "On the detection of applications in co-resident virtual machines via a memory deduplication side-channel," *ACM SIGAPP Appl. Comput. Rev.*, vol. 18, no. 4, pp. 31–46, Jan. 2019.

[70] N. Gattu, M. N. I. Khan, A. De, and S. Ghosh, "Power side channel attack analysis and detection," in *Proc. 39th Int. Conf. Comput.-Aided Design*, 2020, pp. 1–7.

[71] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: GPU side channel attacks are practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 2139–2153.

[72] G. Saileshwar, C. W. Fletcher, and M. Qureshi, "Streamline: A fast, flushless cache covert-channel attack by enabling asynchronous collusion," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2021, pp. 1077–1090.

[73] H. Oh, A. Ahmad, S. Park, B. Lee, and Y. Paek, "TRUSTORE: Side-channel resistant storage for SGX using Intel hybrid CPU-FPGA," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1903–1918.

[74] P. Cronin and C. Yang, "A fetching tale: Covert communication with the hardware prefetcher," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 101–110.

[75] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic, "A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 123–138.

[76] R. Kumar, X. Liu, V. Suresh, H. K. Krishnamurthy, S. Satpathy, M. A. Anders, H. Kaul, K. Ravichandran, V. De, and S. K. Mathew, "A time-/frequency-domain side-channel attack resistant AES-128 and RSA-4K crypto-processor in 14-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 56, no. 4, pp. 1141–1151, Apr. 2021.

[77] I. Vougioukas, N. Nikoleris, A. Sandberg, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, "BRB: Mitigating branch predictor side-channels," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 466–477.

[78] B. B. Yilmaz, R. L. Callan, M. Prvulovic, and A. Zajic, "Capacity of the EM covert/side-channel created by the execution of instructions in a processor," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 605–620, Mar. 2018.

[79] T. H. Pham, A. Fell, A. K. Biswas, S.-K. Lam, and N. Veeranna, "CIDPro: Custom instructions for dynamic program diversification," in *Proc. 28th Int. Conf. Field Program. Logic Appl.(FPL)*, 2018, pp. 224–2245.

[80] B. Gulmezoglu, M. S. Inci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross-VM cache attacks on AES," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 3, pp. 211–222, Jul. 2016.

[81] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A defense against cache timing attacks in speculative execution processors," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 974–987.

[82] Z. Wang, S. Peng, W. Jiang, and X. Guo, "Defeating hardware prefetchers in flush+reload side-channel attack," *IEEE Access*, vol. 9, pp. 21251–21257, 2021.

[83] G. Chen, M. Li, F. Zhang, and Y. Zhang, "Defeating speculative-execution attacks on SGX with HyperRace," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Dec. 2019, pp. 1–8.

[84] S. Mane, M. Taha, and P. Schaumont, "Efficient and side-channel-secure block cipher implementation with custom instructions on FPGA," in *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, 2012, pp. 20–25.

[85] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Efficient power-analysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 1, pp. 49–61, Jan. 2014.

[86] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic, "EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 71–85.

[87] M. He, C. Ma, J. Ge, N. Gao, and C. Tu, "Flush-detector: More secure API resistant to flush-based spectre attacks on ARM cortex-A9," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2020, pp. 1–6.

[88] S. Hsiao, Y. S. Sun, and M. C. Chen, "Hardware-assisted MMU redirection for in-guest monitoring and API profiling," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2402–2416, 2020.

[89] M. Tan, J. Wan, Z. Zhou, and Z. Li, "Invisible probe: Timing attacks with PCIe congestion side-channel," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 322–338.

[90] M. K. F. Arsath, V. Ganesan, R. Bodduna, and C. Rebeiro, "PARAM: A microprocessor hardened for power side-channel attack resistance," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 23–34.

[91] X. Wu, Y. He, Q. Zhou, H. Ma, L. He, W. Wang, and L. Chen, "Partial-SMT: Core-scheduling protection against SMT contention-based attacks," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 2020, pp. 378–385.

[92] M. Busi, J. Noorman, J. V. Bulck, L. Galletta, P. Degano, J. T. Muhlberg, and F. Piessens, "Provably secure isolation for interruptible enclaved execution on small microprocessors," in *Proc. IEEE 33rd Comput. Secur. Found. Symp. (CSF)*, Jun. 2020, pp. 262–276.

[93] Y. Mao, V. Migliore, and V. Nicomette, "REHAD: Using low-frequency reconfigurable hardware for cache side-channel attacks detection," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Sep. 2020, pp. 704–709.

[94] T. Kim and Y. Shin, "Reinforcing meltdown attack by using a return stack buffer," *IEEE Access*, vol. 7, pp. 186065–186077, 2019.

[95] A. Barenghi and G. Pelosi, "Side-channel security of superscalar CPUs: Evaluating the impact of micro-architectural features," in *Proc. 55th Annu. Design Automat. Conf.*, 2018, pp. 1–6.

[96] F. Bache, T. Schneider, A. Moradi, and T. Giineysu, "SPARX—A side-channel protected processor for ARX-based cryptography," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2017, pp. 990–995.

[97] P. Kiaei and P. Schaumont, "Synthesis of parallel synchronous software," *IEEE Embedded Syst. Lett.*, vol. 13, no. 1, pp. 17–20, Mar. 2021.

[98] N. K. Patel, P. Krishnamurthy, H. Amrouch, J. Henkel, M. Shamouilian, R. Karri, and F. Khorrami, "Towards a new thermal monitoring based framework for embedded CPS device security," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 524–536, Feb. 2020.

[99] V. Cristiani, M. Lecomte, and T. Hiscock, "A bit-level approach to side channel based disassembling," in *Smart Card Research and Advanced Applications: 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11–13, 2019, Revised Selected Papers 18*. Springer, 2020, pp. 143–158.

[100] B. Semal, K. Markantonakis, R. N. Akram, and J. Kalbantner, "A study on microarchitectural covert channel vulnerabilities in infrastructure-as-a-service," in *Proc. Appl. Cryptogr. Network Secur. Workshops*, vol. 12418, 2020, pp. 360–377.

[101] O. Aciiçmez and W. Schindler, "A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL," in *Topics in Cryptology—CT-RSA 2008: The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8–11, 2008. Proceedings*. Berlin, Germany: Springer, 2008, pp. 256–273.

[102] C. Wang, N. Min-Allah, B. Guan, Y.-Q. Lin, J.-Z. Wu, and Y.-J. Wang, "An efficient approach for mitigating covert storage channel attacks in virtual machines by the anti-detection criterion," *J. Comput. Sci. Technol.*, vol. 34, no. 6, pp. 1351–1365, Nov. 2019.

[103] H. Fang, S. S. Dayapule, F. Yao, M. Doroslovacki, and G. Venkataramani, "Cache-zoomer: On-demand high-resolution cache monitoring for security," *J. Hardw. Syst. Secur.*, vol. 4, no. 3, pp. 180–195, Sep. 2020.

[104] G. Didier and C. Maurice, "Calibration done right: Noiseless flush+flush attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 18th International Conference, DIMVA 2021, Virtual Event, July 14–16, 2021, Proceedings 18*. Springer, 2021, pp. 278–298.

[105] G. Grimsdal, P. Lundgren, C. Vestlund, F. Boeira, and M. Asplund, "Can microkernels mitigate microarchitectural attacks?" in *Secure IT Systems: 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18–20, 2019, Proceedings*. Cham, Switzerland: Springer, Nov. 2019, pp. 238–253.

[106] F. Mosquera, N. Gulur, K. Kavi, G. Mehta, and H. Sun, "CHASM: Security evaluation of cache mapping schemes," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS 2020, Samos, Greece, July 5–9, 2020, Proceedings 20*. Springer, 2020, pp. 245–261.

[107] T. Schlüter and Lemke-Rust, "Differential analysis and fingerprinting of zombieloads on block ciphers," in *Smart Card Research and Advanced Applications: 19th International Conference, CARDIS 2020, Virtual Event, November 18–19, 2020, Revised Selected Papers 19*. Springer, 2021, pp. 151–165.

[108] F. Lang, H. Li, W. Wang, J. Lin, F. Zhang, W. Pan, and Q. Wang, "E-SGX: Effective cache side-channel protection for intel SGX on untrusted OS," in *Information Security and Cryptology: 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11–14, 2020, Revised Selected Papers*. Springer, 2021, pp. 221–243.

[109] R. Bloem, S. Jacobs, and Y. Vizel, "Efficient information-flow verification under speculative execution," in *Automated Technology for Verification and Analysis: 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28–31, 2019, Proceedings 17*. Springer, 2019, pp. 499–514.

[110] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7–8, 2016, Proceedings 13*. Springer, 2016, pp. 279–299.

[111] M. Payer, "HexPADS: A platform to detect 'stealth' attacks," in *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6–8, 2016. Proceedings 8*. Springer, 2016, pp. 138-154.

[112] A. Barenghi, M. Brevi, W. Fornaciari, G. Pelosi, and D. Zoni, "Integrating side channel security in the FPGA hardware design flow," in *Constructive Side-Channel Analysis and Secure Design: 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1–3, 2020, Revised Selected Papers 11*. Springer, 2021, pp. 275–290.

[113] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Abusing Intel SGX to conceal cache attacks," *Cybersecurity*, vol. 3, no. 1, pp. 1–20, Dec. 2020.

[114] A. Moghimi, J. Wichelmann, T. Eisenbarth, and B. Sunar, "MemJam: A false dependency attack against constant-time crypto implementations," *Int. J. Parallel Program.*, vol. 47, no. 4, pp. 538–570, Aug. 2019.

[115] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss, "Netspectre: Read arbitrary memory over network," in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer, 2019, pp. 279–299.
[116] O. Aciiçmez, S. Gueron, and J.-P. Seifert, "New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures," in *Proc. 11th IMA Int. Conf.*, vol. 4887, 2007, pp. 185–203.
[117] O. Aciiçmez, B. B. Brumley, and P. Grabher, "New results on instruction cache attacks," in *Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17–20, 2010. Proceedings 12*. Berlin, Germany: Springer, 2010, pp. 110–124.
[118] J. Gyselinck, J. Van Bulck, F. Piessens, and R. Strackx, "Off-limits: Abusing legacy x86 memory segmentation to spy on enclaved execution," in *Engineering Secure Software and Systems: 10th International Symposium, ESSoS 2018, Paris, France, June 26–27, 2018, Proceedings 10*. Springer, 2018, pp. 44–60.
[119] J. Großschädl, E. Oswald, D. Page, and M. Tunstall, "Side-channel analysis of cryptographic software via early-terminating multiplications," in *Information, Security and Cryptology—ICISC 2009: 12th International Conference, Seoul, Korea, December 2–4, 2009, Revised Selected Papers 12*. Berlin, Germany: Springer, 2010, pp. 176–192.
[120] H. Nemati, P. Buiras, A. Lindner, R. Guanciale, and S. Jacobs, "Validation of abstract side-channel models for computer architectures," in *Computer Aided Verification*, vol. 12224. 2020, pp. 225–248.
[121] M. Brosch, M. Probst, and G. Sigl, "Counteract side-channel analysis of neural networks by shuffling," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1305–1310.
[122] R. Brotzman, D. Zhang, M. T. Kandemir, and G. Tan, "SpecSafe: Detecting cache side channels in a speculative world," *Proc. ACM Program. Lang.*, vol. 5, pp. 1–28, Oct. 2021, doi: 10.1145/3485506.
[123] P. Buiras, H. Nemati, A. Lindner, and R. Guanciale, "Validation of side-channel models via observation refinement," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 578–591, doi: 10.1145/3466752.3480130.
[124] R. Choudhary, J. Yu, C. Fletcher, and A. Morrison, "Speculative privacy tracking (SPT): Leaking information from speculative execution without compromising privacy," in *Proc. MICRO-54: 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 607–622, doi: 10.1145/3466752.3480068.
[125] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack," in *Proc. 49th Annu. Int. Symp. Comput. Archit.* New York, NY, USA: ACM, Jun. 2022, pp. 204–217, doi: 10.1145/3470496.3527416.
[126] P. W. Deutsch, Y. Yang, T. Bourgeat, J. Drean, J. S. Emer, and M. Yan, "DAGguise: Mitigating memory timing side channels," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.* Lausanne Switzerland: ACM, Feb. 2022, pp. 329–343, doi: 10.1145/3503222.3507747.
[127] D. R. Dipta and B. Gulmezoglu, "DF-SCA: Dynamic frequency side channel attacks are practical," in *Proc. 38th Annu. Comput. Secur. Appl. Conf.* Austin TX USA: ACM, Dec. 2022, pp. 841–853, doi: 10.1145/3564625.3567979.
[128] J. Feldtkeller, J. Richter-Brockmann, P. Sasdrich, and T. Güneysu, "CINI MINIS: Domain isolation for fault and combined security," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* Los Angeles CA USA: ACM, Nov. 2022, pp. 1023–1036, doi: 10.1145/3548606.3560614.
[129] H. Fendri, M. Macchetti, J. Perrine, and M. Stojilovic, "A deep-learning approach to side-channel based CPU disassembly at design time," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 670–675.
[130] A. D. Grandmaison, K. Heydemann, and Q. L. Meunier, "ARMISTICE: Microarchitectural leakage modeling for masked software formal verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3733–3744, Nov. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9852775/
[131] J. He, H. Ma, M. Panoff, H. Wang, Y. Zhao, L. Liu, X. Guo, and Y. Jin, "Security oriented design framework for EM side-channel protection in RTL implementations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2421–2434, Aug. 2022.
[132] J. Hur, S. Song, S. Kim, and B. Lee, "SpecDoctor: Differential fuzz testing to find transient execution vulnerabilities," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* Los Angeles CA USA: ACM, Nov. 2022, pp. 1473–1487, doi: 10.1145/3548606.3560578.
[133] A. Ibrahim, H. Nemati, T. Schlüter, N. O. Tippenhauer, and C. Rossow, "Microarchitectural leakage templates and their application to cache-based side channels," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* Los Angeles CA USA: ACM, Nov. 2022, pp. 1489–1503, doi: 10.1145/3548606.3560613.
[134] L. Li, J. Huang, L. Feng, and Z. Wang, "PREFENDER: A prefetching defender against cache side channel attacks as a pretender," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1509–1514.
[135] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* Los Angeles CA USA: ACM, Nov. 2022, pp. 1977–1991, doi: 10.1145/3548606.3560682.
[136] J. Ravichandran, W. T. Na, J. Lang, and M. Yan, "PACMAN: Attacking ARM pointer authentication with speculative execution," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, New York, NY, USA: ACM, Jun. 2022, pp. 685–698, doi: 10.1145/3470496.3527429.
[137] C. Sakalis, S. Kaxiras, and M. Själander, "Delay-on-squash: Stopping microarchitectural replay attacks in their tracks," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, pp. 1–24, Mar. 2023, doi: 10.1145/3563695.
[138] J. P. Thoma and T. Güneysu, "Write me and I'll tell you secrets—Write-after-write effects on Intel CPUs," in *Proc. 25th Int. Symp. Res. Attacks, Intrusions Defenses*. Limassol Cyprus: ACM, Oct. 2022, pp. 72–85, doi: 10.1145/3545948.3545987.
[139] T. Zhang, M. Cai, D. Zhang, and H. Huang, "SeBROP: Blind ROP attacks without returns," *Frontiers Comput. Sci.*, vol. 16, no. 4, Aug. 2022, Art. no. 164818, doi: 10.1007/s11704-021-0342-8.
[140] P. Borrello, D. C. D'Elia, L. Querzoni, and C. Giuffrida, "Constantine: Automatic side-channel resistance using efficient control and data flow linearization," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 715–733, doi: 10.1145/3460120.3484583.

ARSALAN JAVEED received the B.S. degree in telecommunication engineering from the National University of Computer and Emerging Sciences, Islamabad, Pakistan, in 2011, and the M.S. and Ph.D. degrees in computer science and engineering from Sabanci University, Istanbul, Turkey, in 2015, and 2022, respectively. His research interests include software engineering, and testing and security.

CEMAL YILMAZ (Member, IEEE) received the B.S. and M.S. degrees in computer engineering and information science from Bilkent University, Ankara, Turkey, in 1997 and 1999, respectively, and the Ph.D. degree in computer science from the University of Maryland, College Park, MD, USA, in 2005. From 2005 to 2008, he was a Postdoctoral Researcher with the IBM Thomas J. Watson Research Center, Hawthorne, NY, USA. He is currently an Associate Professor of computer science with the Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey. His current research interests include software engineering, software quality assurance, and software security.

ERKAY SAVAS (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Electronics and Communications Engineering Department, Istanbul Technical University, in 1990 and 1994, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering (ECE), Oregon State University, in June 2000. He had worked for various companies and research institutions before he joined Sabanci University, in 2002. His research interests include applied cryptography, data and communication security, privacy in biometrics, security and privacy in data mining applications, embedded systems security, and distributed systems. He is a member of ACM, the IEEE Computer Society, and the International Association of Cryptologic Research (IACR). He is currently an Associate Editor of IEEE TRANSACTIONS ON COMPUTERS and *Journal of Cryptographic Engineering*.