A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography

XIAOXUAN LOU and TIANWEI ZHANG, Nanyang Technological University, Singapore
JUN JIANG, Two Sigma Investments, LP, USA
YINQIAN ZHANG, Southern University of Science and Technology, China

Side-channel attacks have become a severe threat to the confidentiality of computer applications and systems. One popular type of such attacks is the microarchitectural attack, where the adversary exploits the hardware features to break the protection enforced by the operating system and steal the secrets from the program. In this article, we systematize microarchitectural side channels with a focus on attacks and defenses in cryptographic applications. We make three contributions. (1) We survey past research literature to categorize microarchitectural side-channel attacks. Since these are hardware attacks targeting software, we summarize the vulnerable implementations in software, as well as flawed designs in hardware. (2) We identify common strategies to mitigate microarchitectural attacks, from the application, OS, and hardware levels. (3) We conduct a large-scale evaluation on popular cryptographic applications in the real world and analyze the severity, practicality, and impact of side-channel vulnerabilities. This survey is expected to inspire side-channel research community to discover new attacks, and more importantly, propose new defense solutions against them.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Side-channel analysis and countermeasures**; *Cryptanalysis and other attacks;*

Additional Key Words and Phrases: Microarchitecture, cryptography, vulnerability analysis

ACM Reference format:
Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. 2021. A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography. *ACM Comput. Surv.* 54, 6, Article 122 (July 2021), 37 pages.
https://doi.org/10.1145/3456629

1 INTRODUCTION

The history of side-channel attacks dates back to the year of 1996, when Kocher [121] demonstrated that the data leaked from timing channels was sufficient for an attacker to recover the entire secret key. To generalize, vulnerable implementations of cryptographic operations can exhibit secret-dependent non-functional behaviors during the time of execution, which an adversary can observe

This project is supported by the National Research Foundation, Singapore, under its National Cybersecurity R&D Programme (CHFA-GC1-AW03), and NTU Start-up grant.
Authors' addresses: X. Lou and T. Zhang, Nanyang Technological University, Singapore; emails: XIAOXUAN001@e.ntu.edu.sg, tianwei.zhang@ntu.edu.sg; J. Jiang, Two Sigma Investments, LP; email: jiangcj@pathsec.org; Y. Zhang, Southern University of Science and Technology, China; email: yinqianz@acm.org.

0360-0300/2021/07-ART122 $15.00
https://doi.org/10.1145/3456629

and utilize to fully or partially recover sensitive information. Since then, numerous types of side channels (e.g., execution timing [16, 26], acoustic emission [79], electromagnetic radiation [78], and power consumption [47]) have been discovered and exploited to defeat modern cryptographic schemes, allowing adversaries to break strong ciphers in a short period of time with very few trials.

Among these side-channel threats, microarchitectural attacks are particularly dangerous and prevalent. A fundamental cause of such attacks is the conflict between *performance* and *security*. During the evolution of computer architecture, various strategies were introduced to speed up the execution, which may bring side channels that leak the information of applications running on the system. One example is **Simultaneous Multithreading (SMT)**, where multiple threads execute in parallel and share the same CPU core and functional units. This brings not only high performance, but also side channels due to contention for the shared hardware components. Another example is caching: A small hardware component is introduced (e.g., CPU caches, Translation Look-aside Buffer, DRAM row buffer) to store the previously accessed data, which is usually expected to be used again soon due to the principle of locality. Fetching data directly from this component is much faster. However, such timing differences can reveal the victim program's access traces [86, 151, 155].

It is obviously infeasible to disable those features for side-channel mitigation, which can incur tremendous performance overhead. Therefore, effective elimination of side-channel vulnerabilities has been a long-standing goal. Although security-aware cryptographic applications, systems, and architectures were designed to mitigate side-channel attacks, it is, however, still very challenging to remove all side-channel vulnerabilities from the software implementations and hardware designs. As such, the arms race between side-channel attacks and defenses remains heated.

This article provides a comprehensive survey of microarchitectural side-channel attacks and defenses in cryptographic applications. Since we focus on hardware attacks on software, it is necessary to study the vulnerabilities and defense opportunities in both hardware and software levels. We are particularly interested in three questions: (1) *What are the common and distinct features of software vulnerabilities and hardware flaws that lead to side-channel attacks?* (2) *What are the typical mitigation strategies for applications, operating systems, and hardware, respectively?* (3) *What is the status quo of cryptograhpic applications in terms of side-channel vulnerabilities?*

Existing surveys. Past efforts summarized side-channel studies from different perspectives and fail to answer the above questions. First, some works mainly focused on the physical attacks [100, 148, 184], networking attacks [196, 227], or fault injection attacks with integrity breach [66], which have different characteristics or requirements from microarchitectural side-channel attacks. Second, a few surveys [21, 76, 188] only considered the hardware flaws that result in side channels, while ignoring the software vulnerabilities. Third, several efforts focused on vulnerabilities and countermeasures in one certain cryptosystem (e.g., Elliptic Curve Cryptography [13, 70, 71], Pairing-based cryptography [66]). These summaries are outdated due to a large quantity of newly discovered vulnerabilities and implementation improvements afterwards. Fourth, some works only considered specific platforms (e.g., Trusted Execution Environments [172], smart card [195], cloud [18, 196]) or target applications (e.g., key logging [100, 144]), which did not provide comprehensive conclusions.

Our contributions. Our survey has three significant contributions. First, we characterize microarchitectural side-channel attacks comprehensively. We summarize the attack vectors in both hardware designs (Section 3) and software implementations (Section 4). Second, we identify and abstract the key defense strategies, which are categorized into application, system, and hardware, respectively (Section 5). Third, we conduct a large-scale evaluation of mainstream cryptographic applications. We analyze the side-channel vulnerabilities and the corresponding patches in various libraries and products, and we evaluate the severity and impact from a practical perspective

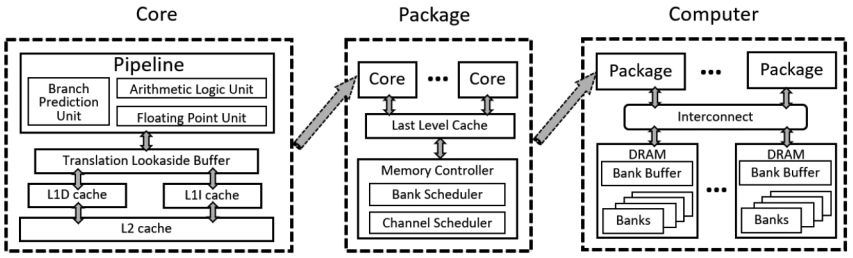


Fig. 1. A Multi-core system.

(Section 6). We hope this work can help researchers, developers, and users better understand the current status and the future direction of side-channel research and countermeasure development.

2 BACKGROUND

2.1 Basics of Side-channel Attacks

In a microarchitectural side-channel attack, the adversary steals secrets by exploiting observable information from the microarchitectural components. Given a secret input $\mathcal{S}$, the target application exhibits certain runtime behaviors $\mathcal{R}$ (e.g., memory access patterns) and causes the underlying host system to reveal some characteristics $\mathcal{I}$. By identifying the correlation $\mathcal{I} \sim \mathcal{R} \sim \mathcal{S}$, the adversary is able to capture the microarchitectural characteristics as the side-channel information and infer the secret input. The success of microarchitectural side-channel attacks relies on vectors from both software and hardware levels.

Software vectors. One necessary condition for microarchitectural attacks is that *application's runtime behaviors need to be correlated with the secrets*: $\mathcal{R} \sim \mathcal{S}$. Generally there are two sources of leakage. (1) *Secret-dependent control flow*: When the secret $\mathcal{S}$ changes, the application executes another code path. (2) *Secret-dependent data flow*: The application may rely on the secret $\mathcal{S}$ to determine the data access location. They yield different behaviors distinguishable by the attacker.

Hardware vectors. The key factor is that *application's behaviors can be reflected by the microarchitectural characteristics*: $\mathcal{I} \sim \mathcal{R}$. Two kinds of techniques exist to capture useful microarchitectural characteristics. (1) An adversary can directly check the states of the hardware component altered by the execution of the application. In this case, the attacker program needs to share the same component with the victim. (2) An adversary can measure the application's execution time to indirectly infer its microarchitectural characteristics. In this case, the attack can be performed without the need to co-locate with the victim, but is only able to capture coarser-grained side-channel information. Thus, a large quantity of sessions are needed to statistically infer useful information.

2.2 Multi-core Architecture

Figure 1 shows the overview of a multi-core system in a hierarchic structure. Basically, a computer has multiple *CPU packages* and *DRAM chips*, interconnected by memory buses (right). Each package is comprised of multiple *CPU cores*, Last Level Caches, and a memory controller (middle). Each CPU core consists of a pipeline, Translation Lookaside Buffer, and two levels of caches (left).

CPU Core. A key feature in modern processors is the multi-stage pipeline, which allows executing instructions in a continuous and parallel manner. The pipeline involves various functional units. For instance, the Branch Prediction Unit predicts the next branch to follow using recently executed branch targets held in the **Branch Target Buffer (BTB)**. The Arithmetic Logic Unit is responsible for the arithmetic and bit-wise operations of integers, while the Floating Point Unit performs

computation on floating point numbers. Modern pipeline designs support **Simultaneous Multi-threading (SMT)**, where multiple threads can execute concurrently in the pipeline. This feature can facilitate side-channel attacks in two ways: (1) the pipeline and functional units are shared among all active threads on the core, and such contention can expose side-channel information. (2) The attacker can measure the victim states concurrently at the same core without interrupting the execution of the victim, and obtain finer-grained information than in the case without SMT.[1]

Processes use virtual addresses for data access, but the memory system uses physical addresses to store the data. Thus, the processor must perform a translation from virtual to physical addresses based on the page table maintained by the operating system. To accelerate the translation, a hardware component named **Translation Lookaside Buffer (TLB)** is introduced to store recent translations, which can be retrieved later at a much higher speed than walking the page table.

CPU caches store recently accessed data for the processor to reuse in the near future and avoid time-consuming main memory access. A cache system is hierarchical and typically consists of three levels. Level 1 (L1) and Level 2 (L2) caches are on-core, while **Last Level Caches (LLCs)** are off-core. Caches closer to the processor are faster to access. There are separate data cache and instruction cache in L1, while L2 and LLC both have mixed data and instruction caches. The smallest storage unit in a cache is a cache line that stores aligned adjacent bytes, which means the processor has to fetch or evict the cache line in its entirety. Modern caches commonly employ the n-way set-associative design, where a cache is divided into multiple sets, each containing n cache lines. A data block is mapped to one cache set determined by its memory address. It can be stored in an arbitrary cache line within this set, selected by a replacement policy. For instance, the **Least Recently Used (LRU)** policy selects the cache line that is least recently accessed to hold the new block when this set is full. Particularly, a cache that has only one way in every set (i.e., $n = 1$) is a direct-mapped cache, while a cache that has only one set is called fully associative.

Package. Each package has one LLC that caches data from applications running on all cores. If a data access request cannot be fulfilled by the LLC, then the memory controller will be involved. The memory controller buffers the requests in multiple queues, schedules them for high performance and fairness, and directs them to the DRAM chips. Cores, the LLC, and the memory controller are interconnected by the memory buses with very high bandwidth.

Computer. A computer consists of several packages and DRAM chips. A DRAM chip has several banks. Each bank can be viewed as a two-dimensional array with multiple rows and columns and has a row buffer to hold the most recently used row to speed up DRAM accesses. A memory access to a DRAM bank may either be served by the row buffer (buffer hit), which is fast, or in the bank itself (buffer miss), which is slow. Packages and DRAM chips are interconnected in a **Non-Uniform Memory Architecture (NUMA)**: Each DRAM is associated with a package, and each package can access all DRAM chips, but it is faster for the package to access its own local DRAM.

Trusted Execution Environment (TEE). This feature protects the security of unprivileged programs from the malicious OS through isolated execution and memory encryption. It has been implemented in ARM TrustZone [11] and Intel SGX [48]. However, as the design of TEE does not consider side-channel attacks, it is possible to use conventional techniques to steal secrets from the protected application. If the attacker is the malicious OS, then she can obtain fine-grained information in an easier way by manipulating the OS interrupt (e.g., SGX-Step [197]). If the attacker is a normal user, then she can use enclaves to hide malicious behaviors [176].

[1]For remote timing attacks, the adversary does not need to launch spy programs on the victim machine, hence SMT does not affect the attack results.

2.3 Cryptography

Asymmetric cryptography. Also known as public key cryptography, it adopts two keys: The user keeps a private key to herself and distributes a public key to the world. This design can provide confidentiality protection: Anyone can use the public key to encrypt a message, which can only be decrypted by the user using the private key. It can also provide digital signature functionality: Given a message, the user can use her private key to generate a signature, which can be verified by anyone using the public key and cannot be forged without the private key. Various algorithms were designed for asymmetric cryptography.

The most famous algorithm is RSA [164]. The key pair is generated using two large prime numbers that are kept secret, and the public key includes their product. The security of RSA relies on the practical difficulty of prime factorization of large numbers. ElGamal [67] is another public-key cryptosystem, defined over any cyclic group, such as the multiplicative group of integers modulo n. Its security is supported by the difficulty of solving the Discrete Logarithm Problem. Yet another approach is **Elliptic Curve Cryptography (ECC)** [137], which is based on the algebraic structure of elliptic curves over finite fields. Its security depends on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem. Schemes based on ECC are designed for digital signature (ECDSA) and key exchange (ECDH).

Symmetric cryptography. It uses the same key for both encryption and decryption, which is shared between two participants and cannot be distributed to the general public. There are generally two types of symmetric-key algorithms. (1) In stream ciphers, each digit of the plaintext is encrypted at a time by a digit from a key stream to produce the ciphertext stream. One common component in stream ciphers is digital shift registers, which generate the key stream from a random seed value. (2) In block ciphers, fixed-length blocks of plaintext bits are blended with the key blocks to generate the ciphertext blocks. The encryption process usually adopts the **Substitution-Perturbation Network (SPN)**, which takes a block of the plaintext and the key as the input and applies multiple alternating rounds of substitution and permutation. AES is the most widely adopted block cipher, which is implemented as a multi-round SPN.

Post-quantum cryptography. The advent of quantum computers in the near future can break the security of classical asymmetric cryptography. As such, post-quantum cryptography, a family of asymmetric ciphers, is proposed to survive attacks by a quantum computer. One popular scheme is lattice-based cryptography. For instance, NTRU [95] utilizes simple polynomial multiplication in the ring of truncated polynomials. **Bimodal Lattice Signature Scheme (BLISS)** [65] provides the digital signature function secure against quantum computers. Other algorithms were proposed based on the **Ring Learning With Errors (RLWE)** hard problem [135].

Cryptographic protocol. SSL/TLS allows a server and a client to use the handshake protocol to exchange a symmetric key K for later secure communications. Specifically, the client first sends a list of its supported cipher suites and the server responds with a list of its supported cipher suites and the server certificate. Then the client picks a cipher (e.g., RSA) supported by both parties and generates a random secret string K as the master key. The client generates a random non-zero padding string pad that is at least 8 bytes, creates a block $0x00||0x02||\text{pad}||0x00||K$, encrypts it using the server's public key, and sends the ciphertext to the server. The server decrypts and accepts the message only when the format is valid. Finally, the server sends a "finished" message to the client and the client replies with a "finished" message, marking the completion of the key exchange.

After the key is established, the server and client adopt CBC-MAC to encrypt messages. The plaintext P is created by concatenating the message m, its **Message Authentication Code (MAC)**, and a padding string chosen to make the byte length of P a multiple of the block size. Then P is

divided into blocks of b bytes, each block encrypted with key K. The final message is the concatenation of a header and all encrypted blocks. The receiver decrypts the ciphertext in CBC mode and validates the padding format and the MAC. If both are correct, then she accepts the original intact message m.

2.4 Threat Model

What we cover. The target of our surveyed works is microarchitectural side-channel attacks. Microarchitecture is defined as the hardware implementation of an **Instruction Set Architecture (ISA)**. We mainly focus on the x86 ISA (e.g., Intel and AMD) due to its wide adoption in modern PCs and servers, although some techniques can also be extended to the ARM processors [87, 128]. Some works may need the processor to have additional hardware features, such as Intel SGX [28, 51, 85, 92, 126, 138, 139, 176, 180, 198, 207, 222], Intel TSX [61, 108], and AMD's cache-way predictor [129]. We will mention the requirements when discussing these works.

We consider the attacker as a normal user in the target system without root privileges. She can launch a malicious program on the same machine as the victim program, but cannot control the scheduling of the attacking process or the victim. One exception is the TEE scenario, where the attacker can be the OS that has the privilege to schedule all processes, but cannot introspect into the victim's protected memory. In remote timing attacks, the attacker can only query the victim cryptographic program remotely without launching the malicious program on the host machine.

What we do not cover. The following attacks and scenarios are out of the scope of this article:

Physical side-channel attacks: These require the attacker to be physically local to the target system to collect the physical signals (e.g., power consumption [47], electromagnetic radiation [78], acoustic emission [79]) during the execution.

Network side-channel attacks: An adversary can exploit the network application features (e.g., response messages, packet pattern, and size) as side channels to attack the network services protected by the cryptographic protocols, including RSA padding oracle attacks [23] and CBC-MAC padding oracle attacks [202]. These network attacks have fundamentally different causes from microarchitectural attacks, and hence are not summarized in this article. Note that we still consider the timing attacks that analyze the information leaked from the microarchitectural states of a remote machine.

Transient execution attacks: Meltdown [130] and Spectre [120] attacks were demonstrated to bypass the protection schemes in OSes, followed by many variants [39, 41, 43, 107, 175, 192]. Although side-channel techniques are used in such attacks as a tool to leak secrets, these attacks target all data in the protected memory region instead of only cryptographic secrets.

Invasive attacks: Following the most conventional microarchitectural side-channel attacks, we assume the attacker can only passively spy the behaviors of the victim, rather than actively compromising the integrity of the victim data. For instance, Rowhammer [118], an inherent vulnerability in modern high-density memory modules, can induce bit flips in the adjacent rows by frequently accessing a memory row. Fault attacks can also be achieved via physical means (e.g., laser injection) [66]. Although such active attacks can break cryptographic ciphers (e.g., RSA [19], AES [233]), we do not elaborate relevant works about Rowhammer [74, 116, 134, 147] and fault attacks [96, 114, 168, 186] in this article. Note that RAMbleed [123] is an exception, as it does not interfere with the victim data.

Attacks against non-cryptographic applications: At the application level, attacks exist to identify keystrokes [182] and application states/activities [185]. At the system level, adversaries may infer host configurations [173] and memory layout information [98]. We do not systematize these attacks.

Table 1. Side-channel Attack Vectors in Hardware

Level	Category	Sharing	Attacks	Requirements
Instruction	Multiply	■	Multiplier unit contention [6, 212]	
	Floating point	■	FPU contention [12]	
	Branch	■	BTB contention [4, 126]	[126] requires Intel SGX
	Micro-operation	■	Port contention [7]	
Cache	Cache set	■: L1 & L2, ●: LLC	Prime-Probe [1, 149, 151, 151, 155, 238] [2, 28, 32, 77, 85, 92, 101, 103, 113, 133] [28, 51, 85, 92, 138, 176] Evict-Time [151], Prime-Abort [61]	[28, 51, 85, 92, 138, 176] requires Intel SGX [61] requires Intel TSX
	Cache line	■: L1 & L2, ●: LLC	Flush-Reload [15, 90, 91, 199, 225] Flush-Flush [89], Reload+Refresh [33] Collide-Probe, Load-Reload[129] LRU state leaking[221]	Requires KSM [129] requires AMD predictor
	Cache bank	■	Bank contention [226], MemJam [139]	[139] requires Intel SGX
Memory Page	Page	■	TLB contention [40, 86]	
		▲	Page Fault/Table Entry [180, 198, 207, 222]	Requires Intel SGX
	DRAM bank row	▲	Row buffer contention [158] Rambleed [123]	

■: sharing the same CPU core[2]; ●: sharing the same package. ▲: sharing the same computer.

3 CHARACTERIZATION OF HARDWARE ATTACK VECTORS

We characterize the attack vectors of side-channel techniques from the level in the computer system and the category of side-channel information, as summarized in Table 1.

3.1 Instruction Level

We first consider the instruction-level attacks, which aim to identify when and what instructions are issued by the victim program. Based on the instruction trace, the adversary can infer the cryptographic secrets. Modern processors normally contain numerous arithmetic or logical functional units to perform designated computation. To launch an instruction-level attack, the adversary must share the same CPU core and the target functional units with the victim process. The contention on these units can leak information of issued instructions from the victim to the adversary.

Multiply instruction. Multiplication is a fundamental operation in cryptographic applications. Hardware multiplier units are implemented in the CPU core to accelerate the computation. Wang et al. [212] demonstrated that processes running on the same core can interfere with the multiplier units, and the adversarial process is able to identify the multiply instruction of the victim based on the timing difference. Aciicmez et al. [6] designed a side-channel attack against the RSA implementation in OpenSSL by distinguishing the multiplications from square operations.

Floating point instruction. Another type of arithmetic operations is computation on floating point numbers. Such operations usually have large internal states, and are accelerated by the **Floating Point Unit (FPU)**. Thus, FPU context switch can cause longer computation time. Additionally, floating point instructions with different operands also have distinguishable execution times, which can leak sensitive information [12]. However, this technique is limited to applications with floating point instructions for critical operations, which are relatively rare in cryptographic applications.

[2]If the SMT is enabled, then the attacker and victim programs only need to share the physical core, instead of the logical core. An attacker in a different logical core from the victim but the same physical core can monitor the victim concurrently without interrupting it. This setting improves the success rate of side-channel attacks and is commonly adopted by these works.

Branch instruction. Given that branch operations widely exist in many applications, speculative execution is introduced to accelerate such operations. The basic idea is to guess a branch path and execute the code in that path. Correct branch prediction saves the wait time for branch condition calculation and can significantly improve the performance, dominating the small overhead due to a misprediction. The speculation is implemented by hardware units, such as **Branch Target Buffer (BTB)**, which records the target addresses of multiple previous branches. The adversary can observe the reduced execution time of the victim, thanks to this technique, and deduce the corresponding operations. Aciicmez et al. [4] demonstrated such an attack against RSA in OpenSSL by selectively evicting entries from the BTB. Similar attacks were realized in the Intel SGX platform [126]. Evtyushkin et al. [69] further exploited the directional branch predictor as a new attack vector to steal secret from an SGX enclave.

Micro-operation. The execution of an instruction can be divided into multiple micro-operations in the CPU pipeline. Contention on the corresponding functional units can also reveal the traces of micro-operations. Aldaya et al. [7] demonstrated a novel side-channel vector exploiting the port contention in the Execution Engine, a built-in component of modern processors with Intel Hyper-Threading technology. The adversary can capture side-channel information derived from port contention with very fine spatial granularity.

3.2 Cache Level

The cache system has become one of the most popular microarchitectural side channels due to its large channel capacity and low attack requirement. According to the granularity of leaked information, these attacks can be further divided into three categories. Below, we briefly discuss the attack techniques and the literatures. Detailed modeling of these attacks can be found in Reference [237].

Cache set. This type of attack aims at identifying the cache set trace of the victim process, with the limitation that different memory accesses mapped to the same cache set cannot be distinguished. There are multiple techniques to achieve this goal. The most common technique is Prime-Probe [151]. The adversary first fills the critical cache set with its own memory lines (Prime). Then the victim executes for a period of time and potentially touches the set. After that, the adversary can measure the access time to those previously loaded memory lines (Probe). A longer access time indicates that the corresponding cache set has been used by the victim. While it is normally observed through cache hits, Reference [32] proposed that the adversary can use cache miss information for better attack efficiency.

Prime-Probe was first adopted to attack the AES encryption on the L1 data cache [149, 151, 155]. Then Aciicmez et al. [1] applied it to L1 instruction cache to check whether certain instructions are executed by the victim. This attack was enhanced in Reference [2], which combines vector quantization and hidden Markov models to monitor each instruction cache set individually. Zhang et al. [238] further explored the attack in the cloud and demonstrated the practicality to steal information across VMs using the Prime-Probe technique.

Researchers shifted the interest from L1 cache to LLC, as the adversary and victim do not need to share the same CPU core. Liu et al. [133] proposed the first Prime-Probe attack on LLC by reverse engineering the cache slice mapping and attacking specific cache sets. Following this work, Kayaalp et al. [113] further relaxed the attack assumptions and achieved higher resolution. Besides that, Inci et al. [101] conducted the Prime-Probe attack on Amazon EC2 and retrieved the RSA key from the co-located instance. Irazoqui et al. [103] used the technique to monitor cache set traces of LLC in both Xen and VMware ESXi hypervisors, recovering the AES key in just a few minutes. This attack technique can also be mounted from a browser with the portable code, e.g., JavaScript, as demonstrated in Reference [77].

Prime-Probe attacks were also applied to the Intel SGX platform, enabling a malicious OS to retrieve secret information from the enclave applications [28, 51, 85, 92, 138]. Since the OS is responsible for process scheduling and interruption, it can easily conduct Prime-Probe side-channel attacks on different levels of caches either synchronously or asynchronously. Besides, the attacker can also use SGX to conceal the cache attacks [176].

Another technique to monitor the cache set access is Evict-Time [151]. At the Evict stage, the adversary fills up one cache set and evicts the victim's memory lines out of the cache. Then at the Time stage, the victim executes certain blocks of code (e.g., encryption of one plaintext) and the corresponding execution time is measured. A long execution time means that the victim has accessed the critical cache sets during the execution and competed for the cache with the adversary.

In addition to timing attacks, Disselkoen et al. [61] proposed the Prime-Abort attack on the Intel **Transactional Memory (TSX)** processors, where the occurrence of aborts is used to infer the victim's access. At the Prime stage, the adversary initiates a TSX transaction for its memory blocks and fills up the target cache sets. When the victim evicts the adversary's block out of the cache, the adversary observes an abort and detects the victim's access.

Cache line. We next consider the attacks that can retrieve information at the granularity of one cache line, typically realized by the Flush-Reload technique. This requires the adversary to share the same memory line with the victim, e.g., via memory deduplication. The adversary first evicts the critical memory lines out of the cache using dedicated instructions (e.g., *clflush*). After a period of time, she reloads these lines into the cache and measures the access time. A shorter time indicates that the memory lines were accessed by the victim and betrays the access trace to the adversary. This attack was first mounted by Gullasch et al. [91] against the AES implementation on the L1 cache. Then Yarom and Falkner [225] adopted this technique on the LLC to monitor the square and multiply operations and steal keys from the RSA implementation. This method was further used to attack other ciphers such as ECDSA [15, 199]. Gruss et al. [90] proposed a cache template attack, which leverages Flush-Reload to automatically build templates and attack critical applications.

A variant of Flush-Reload is Flush-Flush [89], where the Reload operation is replaced by Flush at the second stage. This technique works, as the execution time of Flush can also reflect whether the memory line is in the cache or not. This technique can reduce the activity on the cache and achieve better stealthiness, but has higher error rates due to the noise in the observation.

Cache line states with the replacement policy can also leak side-channel information. Lipp et al. [129] exploited the cache way predictor in the AMD processor to identify the victim's memory accesses with two new techniques: Collide-Probe and Load-Reload. Briongos et al. [33] reverse engineered the cache replacement policies of the Intel processors and then proposed the Reload+Refresh technique to monitor memory accesses in a cache set without evicting the victim's data. Xiong et al. [221] also presented that the LRU states of cache lines can leak information and demonstrated the attacks on both Intel and AMD processors. Bhattacharya et al. [20] discovered that the prefetching state of the cache lines can result in non-constant time encryption, which leaks timing information for the attacker to reveal the key from CLEFIA.

Cache bank. The adversary can even get finer-grained side-channel information than the cache line. A cache line is divided into multiple cache banks. Concurrent requests to the same line but different banks can be served in parallel. However, requests to the same bank would cause a conflict, resulting in observable execution delay. This cache bank conflict can reveal the access pattern of the secret within one cache line. Yarom et al. [226] demonstrated such a side-channel attack on L1 cache targeting RSA in OpenSSL. Moghimi et al. [139] designed a cache attack in the SGX platform, which is based on the false dependency of memory read-after-write (i.e., 4K Aliasing). This creates

a new timing channel, enabling the adversary to observe the memory accesses in the same cache line with different offsets.

3.3 Memory Page Level

The memory page is the smallest unit for memory management in the OS and computer architecture. It is a contiguous and aligned memory block with a specific size, e.g., 4KB. The microarchitectural components responsible for manipulating memory pages can leak side-channel information at the granularity of the page size, which is coarser than that of instruction-level or cache-level attacks, but still allows the adversary to steal secrets from certain applications.

Page. The TLB is an address translation cache, which is similar to CPU caches in terms of timing channels. Gras et al. [86] introduced a TLB-based side-channel attack, where interferences with the TLB are exploited to infer the victim's memory page trace. Canella et al. [40] identified a new attack, which exploits the interactions with the store buffer to steal information of store addresses.

Page faults can also be used as side-channel information to capture the memory accesses [180, 222]. A malicious OS can allocate a restricted number of physical pages to the victim application. When the application needs to access pages not available in the memory, a page fault is triggered and reported by the CPU. The OS is thus able to observe the memory pages the application tries to access. This technique, however, can induce huge performance overhead due to the large number of page faults. Researchers then proposed more advanced attacks [198, 207], where the adversary can infer the accessed pages based on the flags in the page table entries without the need to raise page faults. Moghimi et al. [141] combined the SGX-Step mechanism [197] with the page-fault-based technique to count the number of instructions issued within one page. This can reveal more information (instruction-level) about the victim program inside SGX enclaves for cryptanalysis.

DRAM bank row. Each DRAM bank has a row buffer that caches the recently used DRAM row, which normally contains multiple pages. It accelerates the memory access, but also introduces a timing channel. Pessl et al. [158] designed a DRAM-based attack by reverse engineering the DRAM addressing schemes. This attack is less practical, as it can only recover very coarse-grained information. However, Kwong et al. [123] recently exploited the data-dependent bit flips induced by the Rowhammer [118] to reveal RSA private key stored in the adjacent pages bit-by-bit.

4 CHARACTERIZATION OF SOFTWARE ATTACK VECTORS

We systematically characterize side-channel vulnerabilities from past works based on different operations in different cryptographic algorithms and protocols. Table 2 summarizes the vulnerabilities covered in this article. For each vulnerability, we present the vulnerable operations, causes, and the corresponding attack techniques.

4.1 Asymmetric Cryptography

Modular multiplication. Given three integers x, y, and m, this operation is to calculate $x * y \bmod m$. Both OpenSSL and GnuPG implement two multiplication routines: naive multiplication and Karatsuba multiplication [110]. The selection of the routine is based on the operand size: The naive routine is taken for small multiplicands, while Karatsuba routine is adopted for large ones. Such implementation introduces control-flow side channels about the operands: Karatsuba routine is typically faster than the native routine. An adversary can measure the execution time to infer the sizes of the operands and then recover the secret key [38].

Table 2. Side-channel Vulnerabilities

Category	Operation	Implementation	Application	Cause	Attack Technique	Reference
Asymmetric Cryptography	Modular Multiplication	Basic and Karatsuba multiplication	RSA	■	Remote timing	[38]
	Modular Exponentiation & Scalar Multiplication	Square-and-multiply & Double-and-add	RSA	■	Cache Flush-Reload	[225]
			ElGamal	■	Cache Prime-Probe	[133, 238]
			EdDSA	■	TLB	[86]
		Square-and-Multiply-always & 'Double-and-Add-always	RSA	■	Branch	[63]
			RSA	■	TLB	[86]
		Sliding window	RSA	■	Cache Prime-Probe	[155]
			RSA	■	Cache Flush-Reload	[17]
			ECDSA	■	Cache Flush-Reload	[9, 15, 72, 199]
			ECDSA	■	Execution Port	[7]
			RSA	□	Cache Prime-Probe	[101]
			ElGamal	□	Cache Prime-Probe	[133]
			ECDSA	□	Cache Prime-Probe	[35]
		Fixed window	RSA	□	Cache bank	[226]
		Montgomery ladder	ECDSA	■	Cache Flush-Reload	[224]
			ECDSA	■	Remote timing	[37]
		Branchless montgomery ladder	ECDH	□	Cache Flush-Reload	[179]
			ECDH	■	Cache Flush-Reload	[80]
			ECDH	■	Remote timing	[112]
	Modular Inverse	Binary Extended Euclidean Algorithm	RSA	■	Branch	[3]
			RSA	■	Page Fault	[215]
			RSA	■	Cache Flush-Reload	[8]
Symmetric Cryptography	Substitution-Permutation	Table lookup	MISTY1	□	Remote timing	[193]
			DES	□	Remote timing	[194]
			AES	□	Remote timing	[5, 16, 26]
			CLEFIA	□	Remote timing	[162]
			AES	□	Cache Prime-Probe	[149, 151]
			AES	□	Cache Evict-Time	[151]
			AES	□	Cache Flush-Reload	[91, 105]
	Shift register	Table lookup	eSTREAM	□	Remote timing	[82]
			HC-256	□	Remote timing	[229]
			LFSR	□	Remote timing	[125]
			SNOW 3G	□	Remote timing	[36]
Post Quantum Cryptography	Distribution Sampling	CDT sampling	BLISS	□	Cache Flush-Reload	[34, 157]
		Rejection sampling	BLISS	□	Cache Flush-Reload	[34, 157]
			BLISS	■	Branch	[68, 190]
	Failure Rate Reduction	Error Correcting Code	Ring-LWE	■	Remote timing	[52]
	Message Randomization	Padding-Hash	NTRU	■	Remote timing	[181]
Cryptographic Protocol	RSA-PAD	Uniform response message	TLS	■	Page, Cache, Branch	[220]
			TLS	■	Cache, Branch	[166]
			XML Encryption	■	Cache Flush-Reload	[239]
	CBC-MAC-PAD	Constant-time compression	TLSv1.1, TLSv1.2	■	Cache Flush-Reload	[106]
			TLS	■	Page, Cache, Branch	[220]
			TLS	■	Cache Prime-Probe	[167]

(■: control flow, □: data flow).

Modular exponentiation & scalar multiplication. We consider the two operations together, as they share similar implementations and vulnerabilities. Modular exponentiation is to calculate $x^y \bmod m$, where x, y and m are three integers. Scalar multiplication is to calculate yx where y is a scalar and x is a point on the elliptic curve. The implementations of these two operations can reveal the secret key y in RSA and ElGamal, or the secret scalar y in ECC via side channels.

The first implementation of modular exponentiation is *square-and-multiply* [84], where the calculation is converted into a sequence of `SQUARE` and `MULTIPLY` operations. The binary representation of y is denoted as $y_{n-1}y_{n-2}...y_0$. Then starting from $n-1$ to 0, for each bit y_i, `SQUARE` is called. If y_i is 1, then `MULTIPLY` is also called. Similarly, scalar multiplication adopts the *double-and-add* implementation [93], which runs a sequence of `PointDouble` and `PointAdd` based on each bit y_i. Such implementations are vulnerable to control-flow attacks: The execution of `MULTIPLY` or `PointAdd` depends on bit y_i. By observing the traces of `SQUARE` and `MULTIPLY` in modular

exponentiation, or PointDouble and PointAdd in scalar multiplication, an adversary can fully recover y. In earlier days, this implementation has been attacked via side channels [60, 121]. More recently, successful attacks were demonstrated against RSA in GnuPG via cache Prime-Probe [133, 238] and Flush-Reload [225] attacks, and against EdDSA via TLB attacks [86].

The second implementation of modular exponentiation is *square-and-multiply-always*, which was designed to mitigate the above vulnerability. It always executes both SQUARE and MULTIPLY operations for each bit and selects the output of SQUARE if y_i is 0, or the output of MULTIPLY following SQUARE if y_i is 1. Similarly, *double-and-add-always* [93] was proposed for scalar multiplication in ECC. These implementations execute a fixed number of SQUARE (PointDouble) and MULTIPLY (PointAdd) operations, defeating remote timing attacks. However, output selection still requires a secret-dependent branch, which is usually smaller than one cache line. If it fits within the same cache line with the preceding and succeeding code, then it is not vulnerable to microarchitectural attacks. However, Doychev and Köpf [63] showed that for Libgcrypt, some compiler options can put this branch into separate cache lines, making this implementation vulnerable to cache-based attacks. Gras et al. [86] showed that this branch can be put into separate pages, and the implementation is subject to TLB-based attacks.

The third implementation is *sliding window* [27]. For modular exponentiation, the exponent y is represented as a sequence of windows d_i. Each window starts and ends with bit 1, and the window length cannot exceed a fixed parameter w. So the value of any window is an odd number between 1 and 2^w-1. This method pre-computes $x^v \bmod m$ for each odd value $v \in [1, 2^w-1]$ and stores these results in a table indexed by $i \in [0, (v-1)/2]$. Then it scans every window, squares and multiplies the corresponding entry in the table. Similarly, for scalar multiplication, the scalar y is represented as a ***w*-ary non-adjacent form (*w*NAF)**, with each window value $d_i \in \{0, \pm 1, \pm 3, \ldots, \pm(2^{w-1}-1)\}$. It first pre-computes the values of $\{1, 3, \ldots, 2^{w-1}-1\}x$ and stores them into a table. Then it scans each window, doubles and adds $d_i x$ (in case $d_i < 0$, adding $d_i x$ becomes subtracting $(-d_i)x$).

Two types of vulnerabilities exist in such implementations. The first one is a secret-dependent control flow: Different routines will be called, depending on whether a window is zero. By monitoring the execution trace of those branches, the adversary learns if each window is zero and further recovers the secret. Such attacks have been realized against RSA [17, 155] and ECDSA [7, 9, 15, 72, 199]. The second one is a secret-dependent data flow: The access location in the pre-computed table is determined by each window value. By observing the access pattern, the adversary is able to recover each window value. Attacks exploiting this vulnerability have been mounted on RSA [101], ElGamal [133], and ECDSA [35].

The fourth implementation is *fixed window* [111], designed to approach true constant-time implementation. Similar to sliding window, it also divides the secret y into a set of windows, pre-computes the exponentiation or multiplication of each window value, and stores the results in a table. The differences are that the window size is fixed as w, and the table stores both odd and even (including zero) values. It removes the critical control flow branch at the cost of more memory and slower runtime. To remove the critical data flow, this approach can be combined with scatter-gather memory layout technique [31], which stores the pre-computed values in different cache lines instead of consecutive memory locations. Specifically, each window value is stored across multiple cache lines, and each cache line stores parts of multiple window values. When MULTIPLY or PointAdd is executed, multiple cache lines are fetched to reconstruct the window value, hiding the access pattern from the adversary. Recently, Moghimi et al. [140] performed a black-box timing analysis to steal private keys from the fixed window scalar multiplication inside the Intel **Trusted Platform Module (TPM)**.

This implementation is still vulnerable to attacks [226] using the cache bank, the minimal data access unit in caches. As previously mentioned, the timing difference between hitting the same

bank and hitting different banks in the same cache line enables the adversary to infer the window values accessed during the gathering phase and then recover the secret bits. Garcia et al. [156] discovered a software bug in the DSA implementation in OpenSSL 1.0.2h: The flag to enable the fixed-window exponentiation is not correctly passed to the call site and thus the modular exponentiation still takes the insecure sliding window code path.

The fifth implementation is *masked window*, derived from the fixed window implementation to further hide the cache bank access patterns. The idea is to access all window values instead of just the one needed and then use a mask to filter out unused data. It performs a constant sequence of memory accesses and has been proven secure against different types of cache-based attacks [63].

The sixth implementation, *Montgomery ladder* [109, 145], is a variation of double-and-add-always for scalar multiplication. It also represents y in the binary form and executes both `PointAdd` and `PointDouble` functions for each bit, irrespective of the bit value. The outputs of the functions are assigned to the intermediate variables determined by the bit value. A difference from double-and-add-always is that in Montgomery ladder, the parameter of `PointDouble` is also determined by the bit value. Thus, the implementation contains even more secret-dependent branches. Yarom and Benger [224] adopted cache Flush-Reload technique to identify the branch patterns in an attack to ECDSA in OpenSSL. Brumley and Tuveri [37] discovered that a loop in OpenSSL 0.9.8 begins from the most significant non-zero bit in y to 0. So the number of iterations is proportional to $log(y)$. This presents a vulnerability for remote timing attacks.

The last approach is *Branchless Montgomery ladder* [124], which replaces branches in Montgomery ladder with a function that uses bitwise logic to swap two intermediate values only if the bit is 1, and thus removes the timing channel. However, the implementations of `PointAdd` and `PointDouble` can still bring side channels. First, OpenSSL adopts a lookup table to accelerate the square operation in these two functions. The access pattern to the table can leak information about the secret in ECDH [179]. Second, the modulo operation $x \bmod m$ in these two functions adopted the early exit implementation: x is directly returned if its value is smaller than m. This branch can be exploited by the adversary to check whether x is smaller than m and then deduce secrets in ECDH [80]. Third, Kaufmann et al. [112] discovered that in Microsoft Windows, the multiplication function of two 64-bit integers has an operand-dependent branch: If both operands have their 32 least significant bits equal to 0, then the multiplication is skipped and the result will be 0. This early exit branch was exploited to attack ECDH.

Modular Inverse. This operation is to calculate the integer x^{-1} given x and m such that $x^{-1}x \equiv 1 \bmod m$. It can also be used to check if two integers are co-prime. One possible implementation is ***Binary Extended Euclidean Algorithm* (BEEA)** [111], which uses arithmetic shift, comparison, and subtraction to replace division. It is particularly efficient for big integers, but suffers from control flow vulnerabilities due to the introduction of operand-dependent branches. Branch prediction [3] attacks were demonstrated to recover the value of m in ECDSA and RSA. Page fault [215] and cache Flush-Reload [8] techniques were adopted to attack the *gcd* operation in RSA key generation.

An alternative approach is ***Extended Euclidean Algorithm* (EEA)**. It calculates quotients and remainders in each step without introducing secret-dependent branches. It is less efficient but secure against control flow side-channel attacks. Currently there are no side-channel vulnerabilities discovered in this implementation. However, some cryptographic libraries have software bugs that may disable this implementation accidentally. García and Brumley [75] discovered that in OpenSSL 1.0.1u, the constant-time flag in ECDSA is not set for the secret nonces. Thus, the modular inverse computation still calls the vulnerable BEEA function instead of the secure EEA one.

4.2 Symmetric Cryptography

Symmetric ciphers, including block ciphers and stream ciphers, are also vulnerable to microarchitectural side-channel attacks. Different from asymmetric ciphers that usually contain lengthy and complex mathematical operations, symmetric ciphers typically leverage lookup tables instead of branch instructions or data-dependent rotations in computation. As a result, symmetric ciphers are more susceptible to data-flow side-channel vulnerabilities than control-flow ones.

Substitution-Permutation. This is a series of linked mathematical operations used in block ciphers. It takes a block of the plaintext and the key as inputs and applies several alternating rounds of substitution boxes and permutation boxes to produce the ciphertext block.

The most common implementation of Substitution-Permutation is table lookup, which converts the algebraic operations in each round into memory accesses. Since the accessed entries of the lookup tables are determined by the secret keys and the plaintexts, an adversary can capture such access patterns to infer secrets. There are generally three types of attacks.

The first one is to steal the keys based on the entire execution time. Tsunoo et al. [193] discovered that the numbers of cache hits and misses when accessing the lookup table can affect the encryption time. Based on this observation, a cache timing attack was mounted on the block cipher MISTY1 [193] and further other ciphers, including DES [194], AES [16] and Camellia [241]. After that, an improved attack technique [26] was designed that leverages the cache access collisions to attack AES in OpenSSL. Aciicmez et al. [5] introduced a realistic remote cache timing attack to steal the AES key by analyzing only the first two encryption rounds.

The second type of attacks is to build "templates" to infer the access pattern of the lookup table during encryption. Accesses to different entries in the lookup table can cause changes in encryption time, and such timing difference is only related to the lookup indexes when the host system configuration is deterministic. Hence, the adversary can construct a template trace of execution time on a system with the same configuration as the victim's and then perform correlation analysis on the trace. Bernstein [16] used a large quantity of plaintexts to construct the timing template and achieved remote cache timing attack on the first round of AES in OpenSSL. A few follow-up studies [42, 150, 152] reproduced the attack and analyzed why it can reveal the key remotely. This technique was further applied to other block ciphers, e.g., CLEFIA [162].

The third type checks the state of the cache during or after the encryption to infer the internal state of the target cipher. Osvik et al. [151] analyzed the cache state after encryption to deduce lookup operations in the first two rounds of AES. They carried out the attacks in both synchronous and asynchronous modes. Neve et al. [149] followed this method to target the last round of AES in OpenSSL and successfully recovered the key with only 14 plaintexts. Recent work tried to sniff the cache state of each table lookup operation to obtain finer-grained side-channel information. Gullasch et al. [91] manipulated the OS scheduler to craft a spy process that steals the cache set address of each table lookup very precisely. Irazoqui et al. [105] adopted the cache Flush-Reload technique to obtain detailed access pattern of the victim cipher.

Shift register. This critical component in stream ciphers is designed to generate pseudorandom key streams. In some stream ciphers, the core logic is implemented as lookup table operation to achieve efficiency. This gives rise to data flow side-channel vulnerabilities.

Gierlichs et al. [82] performed a theoretical analysis on the susceptibility of eSTREAM candidates against side-channel attacks and discovered that some of them are vulnerable from cache timing attacks due to the usage of lookup tables. Zenner [229] described a cache timing attack on the HC-256 stream cipher and offered multiple suggestions for hardening the cipher. He further analyzed eSTREAM finalists and pointed out that most stream ciphers are surprisingly resistant to cache timing attacks as long as the lookup table is not adopted [230]. Leander et al. [125] applied

the cache timing analysis on LFSR-based stream ciphers and proposed a general framework showing that the internal state of these ciphers can be recovered with very little computational effort. On this basis, Brumley et al. [36] presented a cache timing attack on the SNOW 3G stream cipher, recovering the full cipher state in a short time.

4.3 Post-quantum Cryptography

Although post-quantum cryptography is secure against quantum computer based attacks, the implementations of those algorithms may contain side-channel vulnerabilities that are subject to attacks even by a classical computer.

Distribution sampling. This operation is to sample an integer from a distribution. It is essential for BLISS [65] to make the signature statistically independent of the secrets. However, an adversary can adopt side-channel attacks to recover the sampled data, and hence the secrets.

One popular and efficient sampling method is **Cumulative Distribution Table (CDT)** sampling [154], which pre-computes a table $\mathrm{T}[i] = \mathbb{P}[x \leq i | x \sim D_\sigma]$. At the sampling phase, a random number r is uniformly chosen from $[0, 1)$, and the target i is identified from T that satisfies $r \in [T[i-1], T[i])$. Some implementations adopt a guide table I to restrict the search space and accelerate the search process. BLISS adopts this approach to sample blinding values from a discrete Gaussian distribution and add them to the signature. However, the access pattern to the two tables reveals information about the sampled values. An adversary can adopt the cache Flush-Reload technique to recover the blinding values and further the secret key in BLISS [34, 157].

Rejection Sampling [81], alternatively, samples a bit from a Bernoulli distribution $B(\exp(-x/2\sigma^2))$. The implementation can bring side-channel opportunities to steal the secret x: (1) a lookup table $\mathrm{ET}[i] = \exp(-2^i/(2\sigma^2))$ is pre-computed to accelerate the bit sampling, causing a data flow vulnerability; (2) the sampling process needs to iterate over each secret bit and different branches will be executed for different bit values, resulting in a control flow vulnerability.

Practical attacks that exploit those vulnerabilities exist. First, rejection sampling can replace CDT sampling for blinding value generation. An adversary could utilize cache [34, 157] or branch-based [68] attacks to recover the sampled values in BLISS. Second, this approach can also be used to sample random bits to probabilistically determine whether the blinding value is positive or negative and whether the signature should be accepted or rejected. An adversary can infer the secret from this process via cache or branch traces [68, 190].

Failure rate reduction. Post-quantum schemes may have certain failure rate during encryption or decryption due to its statistic nature. Thus, it is necessary to devise mechanisms to reduce the possibility of failure. **Error Correcting Code (ECC)** is adopted to significantly reduce the failure rate, but its implementation can reveal whether the ciphertext contains an error via timing channels: A ciphertext without an error is much faster to decode than one with errors. An adversary can exploit such information to recover the key [52].

Message randomization. Some post-quantum schemes require the message to be randomized during encryption and decryption. This process can also create side-channel vulnerabilities. For instance, encryption and decryption in NTRU use hash functions to randomize the messages. However, the number of hash function invocations highly depends on the input message. As a result, the total execution time of encryption or decryption will vary on different inputs. By measuring such time information, an adversary is able to recover the secret input [181].

4.4 Cryptographic Protocol

In addition to cryptographic algorithms, side-channel attacks were proposed to target cryptographic protocols (specifically, their padding mechanisms).

RSA-PAD. SSL/TLS commonly adopts RSA to exchange the symmetric key K following **Public Key Cryptography Standards (PKCS)**. The client pads the key K, encrypts it using RSA, and sends the ciphertext to the server. When the server receives the ciphertext, she accepts the decrypted message only if the first two bytes are $0x00||0x02$. Otherwise, she sends an error message back to the sender and aborts the connection. This message serves as a side channel to recover the plaintext [23]: When the client sends out a ciphertext, the adversary can intercept the message and send a modified one to the server. From the server's response, the adversary can learn if the first two bytes of the plaintext are $0x00||0x02$ (PKCS conforming) or not. This can reduce the scope of the plaintext. The attacker can repeat this process until the scope is narrowed down to one single value.

A common defense is to unify the responses for valid and invalid paddings: If the decrypted message structure is not PKCS conforming, then the receiver generates a random string as the plaintext and performs all subsequent handshake computations on it. Thus, the adversary cannot distinguish valid ciphertexts from invalid ones based on the responses. However, the adversary can still adopt microarchitectural attack techniques to identify such information. Xiao et al. [220] adopted cache and control inference attacks to identify several control-flow vulnerabilities, which are mainly for improper error logging and reporting mechanisms. Ronen et al. [166] evaluated TLS implementations in several applications and found seven of them were vulnerable to cache Flush-Reload and branch prediction attacks due to secret-dependent control flows in data conversion, padding verification, and padding oracle mitigation. Zhang et al. [239] adopted the cache Flush-Reload technique to capture the access traces as the padding oracle of XML encryption.

CBC-MAC-PAD. Standard network protocols adopt CBC-MAC to encrypt messages. The plaintext P is composed of the message, its **Message Authentication Code (MAC)** and a padding string pad, and is encrypted in CBC mode. The receiver decrypts the ciphertext and validates the padding format and the MAC. If both are correct, then she accepts the original intact message m. If the format is invalid, then she rejects the message with a *decryption_failed* error response. If the format is correct but the MAC is incorrect, then she rejects the message with a *bad_record_mac* error response. These three conditions with three different responses create a side channel [202]: An adversary can modify the ciphertext and send it to the receiver for decryption. Based on the response, he can learn whether the chosen ciphertext is decrypted into an incorrect padding. This oracle enables the adversary to learn each byte of an arbitrary plaintext block.

Different countermeasures were designed to mitigate such side channels: Responses for both invalid padding format error and invalid MAC error are unified to be indistinguishable to the adversary [142]. Dummy MAC validation, padding data, and compression operations were added to make the validation constant-time. However, these implementations still contain secret-dependent control flows, rendering them vulnerable to microarchitectural attacks. An adversary can obtain the padding validation results via cache Flush-Reload [106], Prime-Probe [167], or control-flow inference [220] techniques.

5 SUMMARY OF SIDE-CHANNEL COUNTERMEASURES

In this section, we summarize the potential defenses against microarchitectural attacks and categorize them into three application-level, four system-level, and three hardware-level strategies.

5.1 Application-level Strategies

Runtime behavior unification. Control flow vulnerabilities exist when different secret values lead to different code paths that are distinguishable by the adversary using certain side-channel techniques. Two strategies can be used to remove such control flow.

The first strategy is *always-execute-and-select-by-condition*. All possible code paths are executed regardless of the branching condition. Based on the secret value, the correct result is assigned to the return variable. This technique is adopted in modular exponentiation (square-and-multiply-always), scalar multiplication (double-and-add-always), and CBC-MAC-PAD (constant-time compression). This strategy is effective against remote timing attacks. However, the control flow in result selection can still be observed by a local adversary via microarchitectural attacks. Additionally, if the values for all code paths are pre-computed and stored in memory, then the adversary can also infer the secret via data flow, exemplified by sliding window implementations in modular exponentiation and scalar multiplication. The second strategy is *always-execute-and-select-by-bit*. The difference between this strategy and the previous one is that the selection phase employs bitwise operations of the secret and thus avoids introducing branches or access patterns. The branchless Montgomery ladder algorithm adopts this solution for constant-time conditional swap in scalar multiplication.

Data flow vulnerabilities exist when different values of the secret lead to different memory accesses that can be observed by the adversary. Two strategies can remove such data flow. The first strategy is *always-access-and-select-by-bit*. It accesses all critical locations and selects the correct value based on the bitwise operation. It is adopted in masked window modular exponentiation and scalar multiplication. The second strategy is *calculate-on-the-fly*. We can calculate the value every time it is used instead of pre-computing all values and storing them into a table, particularly when the calculation is inexpensive and does not introduce secret-dependent control flows. Branchless Montgomery ladder adopts this method in the square operation of scalar multiplication.

Runtime behavior randomization. For asymmetric ciphers, one possible approach is cryptographic blinding. There are generally two types of blinding techniques.

We can adopt the *key blinding* technique: A random factor is blended into the secret key, but the original key and the randomized key generate the same cryptographic result. The adversary can only obtain the randomized key via side-channel attacks, which is useless without knowing the blended random factor. For ECDSA and ECDH, the randomized key is $k + sr$, where r is a random number and s is the group order. The scalar multiplication generates $(k + sr)G$, the same as kG [47]. For RSA and ElGamal, the randomized key is $d + r\phi(n)$ where r is a random number and ϕ is the Euler's totient function. The decryption yields $c^{d+r\phi(n)} \bmod n$, which is the same as $c^d \bmod n$. In both cases, the true value of k is hidden from side-channel adversaries.

We can also utilize *plaintext/ciphertext blinding*, which randomizes the plaintexts or ciphertexts that are adaptively chosen by the adversary. The randomized texts cause the adversary to recover a wrong key via side-channel analysis. This solution works only if correct ciphertexts can be produced from randomized plaintexts and vice versa. For ECDSA and ECDH, we can choose a random point R and use $G' = G + R$ in computation. The adversary cannot recover k from the side-channel observation without the knowledge of R [47], but we can easily reproduce the correct result kG by subtracting kR from kG'. For RSA and ElGamal, we can generate a random value r, and replace c with $c * r^e$. Now the decryption process is randomized to be $(c * r^e)^d \bmod n = c^d * r^{ed} \bmod n$. To get $c^d \bmod n$, we can simply multiply the result by r^{-1}, as $r^{ed} * r^{-1} \equiv 1 \bmod n$.

For symmetric ciphers, the main target of side-channel attacks is the lookup table. One simple mitigation idea is to periodically randomize the entries in the table at runtime. Brickell et al. [30] designed compact and frequently randomized S-box for AES. Although the implementation is nearly 2× slower, it can indeed defeat the cache timing attack proposed in Reference [16].

Software vulnerability identification. Some static approaches were designed to identify or verify potential side-channel vulnerabilities in commodity software. Abstract interpretation is a common approach to analyze the source code and measure the information leakage (bounds).

CacheAudit [63, 64] modeled the relationship between the adversary's observation and program's execution traces as a Markov chain and quantified the upper bound of the adversary's probability of success and the information leakage. Molnar et al. [143] modeled control-flow side channels with a program counter transcript, in which the value of the program counter at each step is leaked to an adversary. FlowTracker [165] adopted information flow tracking to analyze the assembly instructions and identify the implicit flow edges in constant-time implementation of Elliptic Curve Cryptography. Irazoqui et al. [104] designed a static code analysis tool to look up implicit features of microarchitectural attacks. SC-Eliminator [219] eliminated side-channel leakage using program repair, which conducts code transformations on unbalanced conditional jumps and cross cache line memory accesses to equalize the execution time. Wang et al. [205] proposed Secret-Augmented Symbolic domain to track program secrets and their dependencies for precision and coarse-grained public information for scalability. Various approaches were proposed [10, 14, 22, 25, 53, 122, 163] to verify constant-time behavior of a program and check if it has secret-dependent conditional jumps or memory accesses.

There are also some dynamic analysis approaches that focus on concrete program executions and identify vulnerabilities from runtime execution traces. Zankl et al. [228] profiled the number of executions in a modular exponentiation operation and calculated the Pearson correlation coefficient between this number and the Hamming weights of the exponent to identify information leakage during modular exponentiation. Wang et al. [206] proposed CacheD to identify the vulnerable instructions by feeding different secrets into the program and checking if each instruction has memory accesses to different cache locations. Shin et al. [179] used the Flush-Reload technique to collect two cache activity traces with two different secret inputs and applied K-means clustering algorithm to check the dependency between cache activities and secret inputs. Mutual information was adopted [102, 218] to measure the side-channel information leakage and the relationship between secret inputs and memory activities. Weiser et al. [216] exploited Intel Pin tool to collect execution addresses and applied statistic Kuiper's test and Randomized Dependence Coefficient to discover vulnerabilities at the granularity of byte addresses. Xiao et al. [220] proposed a framework to identify padding oracle attacks in SSL/TLS protocols in SGX secure enclaves.

5.2 System-level Strategies

Process execution partitioning. Since one enabling factor of microarchitectural attacks is the shared hardware components, we can enforce resource isolation for each process. The strategy is spatial partitioning, i.e., assigning different parts of the hardware units to processes. For instance, in the cloud scenario, hypervisor-based solutions were designed to defeat LLC attacks by partitioning the LLC, via page coloring [177], page locking [117], and Intel Cache Allocation Technology [177]. Zhou et al. [243] prevented Flush-Reload attacks by managing dynamic page mapping to avoid cache line sharing and thwarted Prime-Probe attacks through reduction of cross-domain cache line eviction. Hardware Transactional Memory was leveraged to eliminate the cache interference and prevent the adversary from evicting the victim's memory lines out of the cache [44, 88]. To defeat side-channel attacks in web browsers, Schwarz et al. [174] designed a fine-grained permission system to restrict the behaviors of JavaScript interface and functions.

Process scheduling. Since a lot of microarchitectural side-channel attacks require the attacker and victim programs to run concurrently on the same machine, one possible strategy is to carefully schedule different programs to achieve temporal partitioning. Zhang and Reiter [240] introduced an OS-based solution, which frequently flushes the local microarchitectural states (BTB, TLB, caches) to reduce side-channel leakage during context switches. Similar ideas were proposed in References [83, 200], where CPU caches are flushed during VM switches to defeat cache

side-channel attacks in the cloud. To reduce the overhead of state cleansing operations, Sprabery et al. [183] implemented the scheduling as an extension to the Completely-Fair-Scheduler in Linux.

Measurement randomization. This idea is to add randomization to the adversary's measurements, making it difficult or infeasible to capture accurate information based on the observations. This was first proposed in Reference [97] to fuzz the timing information to reduce timing channels. Vattikonda et al. [201] modified the *rdtsc* instruction from the hypervisor to randomize the emulated timer. Martin et al. [136] optimized this approach by adding random noise in each predefined epoch. Li et al. [127] introduced Stopwatch, which disables precise timing measurement in the cloud server to mitigate timing-channel attacks.

An alternative way is to add randomization inside the application during compiling. Crane et al. [50] designed an approach to dynamically randomize the control flow in the application to defeat cache side-channel attacks. Braun et al. [29] inserted random temporal paddings into the source application to obfuscate the adversary's observations.

Attack Detection. In addition to prevent side-channel attacks, another direction is to detect the occurrence of side-channel attacks at runtime. The key insight is that the victim and attacking programs involved in a microarchitectural side-channel attack exhibit unusual behaviors compared to normal applications, which can be observed by the privileged software. For instance, signature-based detection systems [46, 55, 153] were proposed to detect side-channel attacks using hardware performance counters. NIGHTs-WATCH [146] leveraged machine learning techniques to detect cache attacks based on the performance counter values. Zhang et al. [236] combined both signature-based and anomaly-based detection methods to identify cache Prime-Probe and Flush-Reload attacks with high fidelity. Hunger et al. [99] proposed mimicking the behaviors of the victim to attract and identify the adversary by monitoring its common characteristics. To detect page-level side-channel attacks in Intel SGX, multiple techniques [45, 178, 187] took advantage of the Intel TSX feature.

5.3 Architecture-level Strategies

Hardware resource partitioning. Computer architects and researchers have designed security-aware architectures to protect critical applications from side-channel attacks. One straightforward way is to partition the shared resources to prevent processes from interfering with one another.

As the CPU cache is the most popular target for microarchitectural attacks, a lot of efforts have been spent to enhance the security of cache architectures. The simplest way is to divide the cache into multiple partitions by ways and allocate them to different processes exclusively, such as Statically Partitioned cache [94] and SecVerilog cache [231, 232]. However, statically partitioned caches can significantly reduce the effective cache capacity for each process, causing huge performance degradation and unfairness.

A more promising direction is dynamic partitioning. Partition-Locked cache [213] assigns a protection bit to each memory line to denote whether it needs to be locked in the cache. Green et al. [87] identified an undocumented feature *AutoLock* on ARM processors to prevent evictions of lines in core-private caches. Vantage [170] enforces fine-grained partitioning during replacement instead of physically restricting the placement of cache lines. NoMo Cache [62] reserves certain blocks in every cache set for each thread, which cannot be evicted by other threads. The number of those blocks is dynamically adjusted based on the activity of the thread. To maintain high associativity while partitioning the cache, Futility Scaling [204] keeps evicting cache lines with the largest scaled futility to retain a number of useful lines. SecDCP Cache [209] improved over SecVerilog cache by dynamically partitioning the cache for different processes based on the cache miss rate

of instructions at runtime. The work was further enhanced in FairSDP [171], which improves the fairness among competing threads. SHARP Cache [223] implements Core Valid Bits to cache lines, enabling the OS to prioritize the cache lines for eviction when cache conflict occurs. This can reduce the interference among different processes. DAWG [119] introduces minimal modifications on the hardware to fully isolate cache his/misses and metadata updates across protection domains in the cache set. ZBM [169] modifies the replacement policy to equalize the latencies of cache hits and misses on certain lines invalidated due to flush-caused invalidation.

This partitioning strategy is also adopted by other platforms and scenarios. To mitigate side-channel attacks in Trusted Execution Environment, Sanctum Cache [49] assigns different memory regions to different enclaves or OSes to disable cache sharing and flushes the caches during context switching from the enclave mode to the non-enclave mode. Following this work, HybCache [59] selectively applies partitioning only for isolated execution domains, making the sharing of cache resources more flexible and efficient.

For other microarchitectural components, Static-Partition TLB [58] was introduced, which follows the idea of static-partitioning cache and Sanctum to isolate the TLB accesses between the victim and the attacker. Wang et al. [210] designed approaches to prevent information leakage via on-chip networks by restricting low-security traffic. To protect the memory controllers, Wang et al. [208] adopted temporal partitioning to group memory access requests in queues according to their security domains and separate the requests serving different domains in different time slots. This design was further improved by the lattice priority scheduling [73] and quantitative security guarantee [211] to reduce performance overhead and increased scalability.

Resource usage randomization. This strategy is to randomize the resource usage of the processes to obfuscate the side-channel observation. Random Permutation cache [213] maintains a dynamic and random memory-to-cache mapping table for each process to ensure the accessed set is unpredictable. Newcache [132, 214] introduced a virtual **Logical Direct-Mapped (LDM)** Cache with two mappings: The one from memory addresses to the LDM cache is direct-mapped for high performance, while the other from the LDM cache to the physical cache is fully associative for strong security. Non Deterministic Cache [115] leverages cache access delay to obscure the relationship between cache accesses and the observed timing information. Random Fill Cache [131] randomizes cache prefetching by bringing the accessed memory line along with random neighbors to the cache, so the memory access pattern is convoluted. TSCache [191] makes the cache access pattern and timing leakage unpredictable in the embedded devices with injection of randomized cache timing behaviors. CEASER Cache [160] encrypts and remaps the addresses of memory lines, which can efficiently decrease the probability of conflict caused by cache misses. An improved version, CEASER-S [161] was then designed to divide the cache into multi-way partitions and adopt random placement among these partitions, further increasing the uncertainty in memory-to-cache mapping. SCATTER Cache [217] translates the memory address and process information to a random cache set index and guarantees that each cache way has its own specific index. Phantom Cache [189] proposed a novel localized randomization method, which randomly places a loaded memory block at a location in its fixed mapping range. Purnal et al. [159] introduced a generic model for randomization-based caches with systematic analysis, which can also serve as a baseline for future cache design.

Randomization can be applied to other microarchitectural components as well. Random-Fill TLB [58] decorrelates the memory access from the actual TLB entry by randomizing the address translation for TLB misses, which can result in non-deterministic observations. Camouflage [242] hides the timing information by shaping the time of memory operations into a predetermined distribution and adding fake memory traffic as needed.

Hardware vulnerability identification. Multiple approaches were designed to model security-aware hardware architectures and measure their vulnerabilities under different attacks. Demme et al. [54] proposed the **Side-channel Vulnerability Factor (SVF)**, a metric to quantify the system's vulnerabilities by measuring the correlation between the attacker's observations and the victim's actual execution. References [20, 235] introduced improved metrics over SVF to assess other hardware components and attacks. Zhang et al. [234] modeled the cache architectures as finite-state machines and then quantitatively revealed potential side-channel leakage. He et al. [94] used probabilistic information flow graph to model the interaction of the attacker and the victim in the cache architecture and measure the cache's resilience against side channels. More analyses of secure cache architectures have been done using computation tree logic [56], three-step model [57], attack graphs [203], and neural networks [237].

5.4 Discussions of These Defenses

Among aforementioned three categories of defense strategies, the application-level ones are the most widely adopted in the cryptographic community due to two reasons: (1) they are easy to implement and patch the vulnerabilities immediately; (2) these approaches bring little computational overhead to the applications. However, these solutions are not very general, and the designs require manual analysis by experts. Hence, it is difficult to guarantee modern applications are free of side-channel vulnerabilities inside the tremendous amount of code. We will perform a comprehensive study about two popular cryptographic applications in the next section.

There are relatively fewer system-level or architecture-level strategies applied to real-world platforms or products, although a lot of general solutions have been well developed in academia. These solutions may have performance issues and are much harder to implement in the existing machines, especially for the new hardware designs. So, currently the most practical defense solutions for side-channel attacks are still based on modifying cryptographic algorithms and implementations.

6 EVALUATION OF CRYPTOGRAPHIC LIBRARIES

From a practical perspective, we review, analyze, and evaluate the development of side-channel attacks and defenses in two commonly used cryptographic libraries: OpenSSL and GNU Crypto (GnuPG, Libgcrypt, and GnuTLS). We collected the history of side-channel vulnerabilities and countermeasures (1999–2019) from **Common Vulnerabilities and Exposures (CVE)**, the version control history and the source code. Tables 3 and 4 show the evolution of the libraries.

6.1 Vulnerability Severity

We examine the severity and practicality of side-channel attacks as well as the attention developers paid to them. We establish the measurements for these threats and compare them with other vulnerability categories. We adopt the **Common Vulnerability Scoring System (CVSS)**[3] to assess each CVE. CVSS is a widely accepted industry standard to identify and assess vulnerabilities across diverse platforms. It contains three metric groups: *Base*, *Temporal*, and *Environmental*, each consisting of a set of metrics. We consider the *Base* score that well represents the inherent quality of a vulnerability. It comprises two sub-scores: *Exploitability,* which defines the difficulty to attack the software; and *Impact,* which defines the level of damage to certain properties of the software under a successful attack. The score ranges from 0 (least severe) to 10 (most severe). Detailed computation of those scores can be found in the **National Vulnerability Database (NVD)** website.[4]

[3]The latest CVSS version is v3.0. We adopt CVSS v2.0, as old vulnerabilities were not assigned CVSS v3.0 scores.
[4]https://nvd.nist.gov/vuln-metrics/cvss/v2-calculator.

Table 3. Vulnerabilities in OpenSSL

ID	Patch Date	Version	Operations	Implementation	CVE		CVSS			Countermeasures
					Date	ID	B	E	I	
1	2001/07/09	0.9.6b	RSA-PAD	Uniform error message						Fix bugs
2	2003/02/19	0.9.6i, 0.9.7a	CBC-MAC-PAD	Uniform error message	2003/03/03	2003-0078	5	10	2.9	Dummy checking for TLS
3	2003/04/10	0.9.6j, 0.9.7b	Modular multiplication	Basic and Karatsuba multiplication	2003/03/31	2003-0147	5	10	2.9	RSA blinding
			RSA-PAD	Uniform error message	2003/03/24	2003-0131	5	10	2.9	Uniform version error message
4	2005/07/05	0.9.8	Modular exponentiation	Sliding window						Fixed window
5	2005/10/11	0.9.7h								
6	2007/10/11	0.9.8f	Modular inversion	Binary Extended Euclidean Algorithm						Euclidean Extended Algorithm
7	2011/09/06	1.0.0e	Scalar multiplication	Montgomery ladder	2011/05/31	2011-1945	2.6	4.9	2.9	Make the bit length of scalar constant
		0.9.8s								
8	2012/01/04	0.9.8s, 1.0.0f	CBC-MAC-PAD	Uniform error message	2012/01/05	2011-4108	4.3	8.6	2.9	Dummy checking for DTLS
				Padding data initialization	2012/01/05	2011-4576	5	10	2.9	Fix bugs
9	2012/03/12	0.9.8u, 1.0.0h	RSA-PAD (PKCS#7,CMS)	Error message	2012/03/12	2012-0884	5	10	2.9	Uniform error message and dummy checking
10	2012/03/14	1.0.1	Scalar multiplication	Sliding window						Masked window
			Substitution-Permutation	T-box lookup						AES-NI support
11	2013/02/05	0.9.8y, 1.0.0k, 1.0.1d	CBC-MAC-PAD	Dummy MAC checking	2013/02/08	2013-0169	2.6	4.9	2.9	Dummy data padding
12	2014/04/07	1.0.1g	Scalar multiplication	Montgomery ladder	2014/03/25	2014-0076	1.9	3.4	2.9	Branchless Montgomery ladder
13	2014/06/05	0.9.8za, 1.0.0m								
14	2014/10/15	0.9.8zc, 1.0.0o, 1.0.1j	CBC-MAC-PAD	Error message	2014/10/14	2014-3566	4.3	8.6	2.9	Disable fallback of SSLv3.0
15	2016/01/28	1.0.1r, 1.0.2f	RSA-PAD	Uniform error message	2016/02/14	2015-3197	4.3	8.6	2.9	Disable SSLv2 ciphers
					2016/03/02	2016-0703	4.3	8.6	2.9	
					2016/03/02	2016-0704	4.3	8.6	2.9	
					2016/03/01	2016-0800	4.3	8.6	2.9	
16	2016/03/01	1.0.1s, 1.0.2g	RSA-PAD	Uniform error message	2016/02/14	2015-3197	4.3	8.6	2.9	Disable SSLv2 protocols
					2016/03/02	2016-0703	4.3	8.6	2.9	
					2016/03/02	2016-0704	4.3	8.6	2.9	
					2016/03/01	2016-0800	4.3	8.6	2.9	
			Modular exponentiation	Fixed window	2016/03/03	2016-0702	1.9	3.4	2.9	Masked window
17	2016/05/03	1.0.1t, 1.0.2h	CBC-MAC-PAD (AES-NI)	Dummy data padding	2016/05/04	2016-2107	2.6	4.9	2.9	Fix bugs
18	2016/09/22	1.0.1u, 1.0.2i	Modular exponentiation	Fixed window	2016/06/19	2016-2178	2.1	3.9	2.9	Fix bugs
19	2018/08/14	1.0.2p, 1.1.0i	Scalar multiplication	Branchless Montgomery ladder						On-the-fly calculation to replace lookup table
			Modular inversion	Binary Greatest Common Divisor	2018/04/16	2018-0737	4.3	8.6	2.9	Extended Euclidean Algorithm
			Modulo	Early exit						ECDSA and DSA blinding
		1.1.0i	Scalar multiplication	Sliding window						Branchless Montgomery ladder
20	2018/09/11	1.1.1	Scalar multiplication	Branchless Montgomery ladder						Differential addition-and-doubling
										Coordinate blinding
				Masked window						Branchless Montgomery ladder
				Sliding window						Branchless Montgomery ladder
			Modular inversion	Extended Euclidean Algorithm						New constant-time function for EC
										Input blinding
21	2018/11/20	1.0.2q	Scalar multiplication	Sliding window	2018/11/15	2018-5407	1.9	3.4	2.9	Branchless Montgomery ladder
		1.0.2q, 1.1.0j,	DSA sign setup	Space preallocation	2018/10/30	2018-0734	4.3	8.6	2.9	Fix bugs
		1.1.1a	Scalar multiplication	Space preallocation	2018/10/29	2018-0735	4.3	8.6	2.9	Fix bugs
22	2019/02/26	1.0.2r	CBC-MAC-PAD	Protocol error handling	2019/02/27	2019-1559	4.3	8.6	2.9	Fix bugs
		1.1.1b	Modular inversion (EC)	Binary Extended Euclidean Algorithm						EC-specific inversion function with input blinding
23	2019/09/10	1.1.1d, 1.1.0l, 1.0.2t	EC Group	Cofactor	2019/09/10	2019-1547	1.9	3.4	2.9	Fix bugs
			RSA-PAD	CMS and PKCS7 decrypt	2019/09/10	2019-1563	4.3	8.6	2.9	Fix bugs

(For CVSS Column, B: Base; E: Exploitability; I: Impact).

The score of each side-channel vulnerability is collected from the NVD[5] and CVE[6] websites. It is worth noting that CVSS is a general metric for characterizing software vulnerabilities. It does not contain evaluation criteria for microarchitectural threats and may not comprehensively reveal the inherent features of microarchitectural attacks. However, it can reflect the attitude of the cryptographic community towards side-channel attacks in a practical way.

For OpenSSL and GNU Crypto, the top vulnerabilities are denial-of-service, arbitrary code execution, buffer overflow, and memory corruption. Table 5(a) compares the average scores and

[5]https://nvd.nist.gov/.
[6]https://cve.mitre.org/.

Table 4. Vulnerabilities in GNU Crypto

ID	Patch Date	Version	Operations	Implementation	CVE		CVSS			Countermeasures
					Date	ID	B	E	I	
1	2006/09/08	T1.4.3	RSA-PAD	Error Message						Uniform error message
2	2006/09/21	T1.5.1								
3	2011/06/29	L1.5.0	Substitution-permutation	T-box lookup						AES-NI support
4	2012/01/06	T3.0.11	CBC-MAC-PAD	Uniform error message	2012/01/05	2012-0390	4.3	4.9	2.9	Dummy checking for DTLS
5	2013/02/04	T2.12.23, T3.0.28, T3.1.7	CBC-MAC-PAD	Dummy MAC checking	2013/05/19	2013-1619	4.3	4.9	2.9	Dummy data padding
6	2013/07/25	P1.4.14, L1.5.3	Modular exponentiation	Square-and-Multiply	2013/08/19	2013-4242	1.9	3.4	2.9	Square-and-Multiply-always
7	2013/12/16	L1.6.0	Modular exponentiation	Square-and-Multiply	2013/08/19	2013-4242	1.9	3.4	2.9	Square-and-Multiply-always
8	2013/12/18	P1.4.16	Modular multiplication	Basic and Karatsuba multiplication	2013/12/20	2013-4576	2.1	3.9	2.9	Exponentiation blinding
9	2014/08/07	L1.5.4	Modular multiplication	Basic and Karatsuba multiplication	2014/10/19	2014-5270	2.1	3.9	2.9	Exponentiation blinding
10	2015/02/27	P1.4.19, L1.6.3	Modular multiplication	Basic and Karatsuba multiplication	2015/02/27	2014-3591	1.9	3.4	2.9	ElGamal Blinding
			Modular exponentiation	Sliding window	2015/02/27	2015-0837	4.3	8.6	2.9	Remove control flow
11	2016/02/09	L1.6.5	Scalar multiplication	Sliding window	2016/04/19	2015-7511	1.9	3.4	2.9	Double-and-Add-always
12	2016/02/18	L1.5.5								
			Modular multiplication	Basic and Karatsuba multiplication	2015/02/27	2014-3591	1.9	3.4	2.9	ElGamal Blinding
13	2016/04/15	L1.7.0	Modular exponentiation	Sliding window	2015/02/27	2015-0837	4.3	8.6	2.9	Remove control flow
			Scalar multiplication	Sliding window	2016/04/19	2015-7511	1.9	3.4	2.9	Double-and-Add-always
14	2017/06/29	L1.7.8	Modular exponentiation	Sliding window	2018/07/26	2017-7526	4.3	8.6	2.9	RSA blinding
15	2017/07/18	L1.8.0								
16	2017/07/19	P1.4.22								
17	2017/08/27	L1.7.9, L1.8.1	Scalar multiplication	Branchless montgomery ladder	2017/08/29	2017-0379	5	10	2.9	Input validation
18	2018/06/13	L1.7.10, L1.8.3	Modulo	Early exit	2018/06/13	2018-0495	1.9	3.4	2.9	ECDSA blinding
19	2018/07/16	T3.3.30, T3.5.19, T3.6.3	CBC-MAC-PAD	Pseudo constant time	2018/08/22	2018-10844	1.9	3.4	2.9	New variant of pseudo constant time (Not fully mitigated)
					2018/08/22	2018-10845	1.9	3.4	2.9	
					2018/08/22	2018-10846	1.9	3.4	2.9	
20	2018/12/01	T3.6.5	RSA-PAD	Pseudo constant time	2018/12/03	2018-16868	1.9	3.4	2.9	Hide access pattern & timing

(For the Version Column, P: GnuPG; L: Libgcrypt; T: GnuTLS. For CVSS Column, B: Base; E: Exploitability; I: Impact).

Table 5. Severity and number of Software Vulnerabilities

	CVSS			Count
	Base	Exploit	Impact	
Denial of Service	5.74	9.46	4.33	157
Buffer Overflow	6.76	9.43	5.77	43
Side channel	**3.32**	**6.27**	**2.90**	**37**
Code Execution	7.98	9.08	7.74	35
Mem Corruption	6.56	9.72	5.29	20

(a) Comparisons with different Vulnerabilities

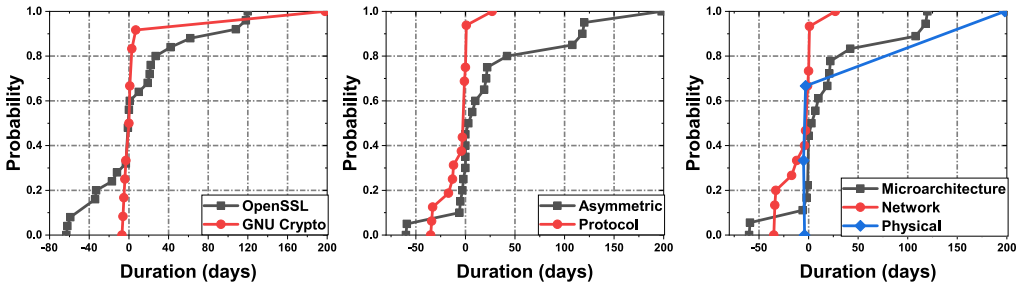
	CVSS			Count
	Base	Exploit	Impact	
Asymmetric Crypto	2.86	5.48	2.90	18
Crypto Protocol	3.76	7.02	2.90	19
Microarchitecture	**3.04**	**5.87**	**2.90**	**16**
Physical	1.97	3.57	2.90	3
Network	3.87	7.22	2.90	18

(b) Comparisons of side-channel vulnerabilities

quantities of these vulnerability categories.[7] We observe that *side-channel vulnerabilities are regarded less severe than other types* due to lower *Exploitability* and *Impact* sub-scores. Side-channel attacks usually require stronger adversarial capabilities, in-depth knowledge about the underlying platforms, and a large amount of attack sessions, but only cause partial confidentiality breach, as they leak (part of) keys or plaintexts. In contrast, other vulnerabilities may enable less experienced attackers to exploit them for executing arbitrary code or disabling the services entirely.

Next, we break down and compare different types of side-channel vulnerabilities, as shown in Table 5(b). We first consider the two categories of operations: asymmetric ciphers and protocol

[7]There are some mistakes in CVEs: (1) all side-channel vulnerabilities should only have partial confidentiality impact, while CVE-2003-0131, CVE-2013-1619, and CVE-2018-16868 were also assigned partial integrity or availability impact. (2) CVE-2018-10844, CVE-2018-10845, and CVE-2018-10846 should have local access vector, but they were assigned network access vector. We corrected them in our analysis.



(a) Different libraries (b) Different operations (c) Different access vectors

Fig. 2. Cumulative distributions of vulnerability windows.

padding. We did not find any side-channel CVEs related to post-quantum cryptography, as its development is still at an early stage. We also skip the vulnerabilities in symmetric ciphers, as there are only two reported CVEs (row 10 in Table 3 and row 3 in Table 4). This is because symmetric ciphers are simpler and thus less vulnerable than asymmetric ones, and the widely adopted AES-NI instruction set extension can effectively mitigate existing vulnerabilities. We observe that *vulnerabilities in protocol padding are generally more severe than those in asymmetric ciphers due to higher Exploitability*. The underlying reason is that *Exploitability* is determined by the access vector: network vector and local vector are neck and neck for vulnerabilities in asymmetric ciphers, but the former dominates access vectors of padding oracle attacks, rendering them more exploitable. We also compare microarchitectural attacks with network and physical attacks. From Table 5(b), we can observe that network attacks are most severe due to high *Exploitability*. Scores of microarchitectural attacks are also higher than that of physical attacks, as some microarchitectural vulnerabilities can be exploited via remote timing measurement, while all physical attacks must be conducted on site.

6.2 Vulnerability Response

We evaluate the responses to discovered side-channel vulnerabilities from application developers.

Response speed. For each vulnerability, we measure the *vulnerability window*, defined as the duration from the vulnerability publication date[8] to the patch release date. If the patch release date is earlier than the vulnerability publication date, then the vulnerability window is negative. Obviously, a narrower vulnerability window (in case of positive) leads to fewer chances of exploit and less damage.

Figure 2(a) shows the cumulative distribution of vulnerability windows for OpenSSL and GNU Crypto. We can see that *both libraries responded to side-channel vulnerabilities very actively*: 56% and 50% of the vulnerabilities were fixed by the two libraries, respectively, before publication; more than 80% of the vulnerabilities were fixed within one month of their disclosure; each library has only one case that spanned more than four months, the longest being 198 days in GnuPG. Figures 2(b) and 2(c) compare the vulnerability windows of different operations and attack types, respectively. *Although network attacks are more severe than local attacks, they were fixed at similar speeds.*

[8]A side-channel vulnerability may be published in different ways, including online archives, formal academic publications, and the CVE system. We use the earliest of all such dates.

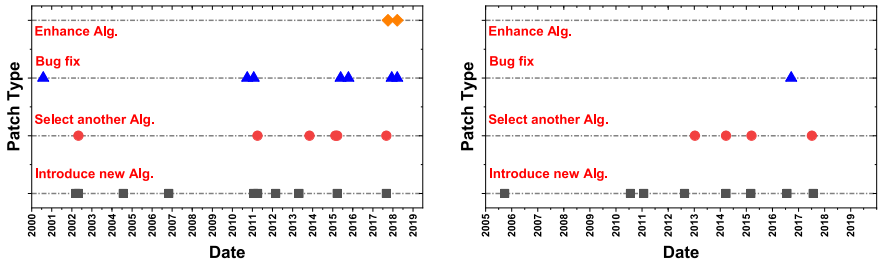
Table 6. Number of Patches for Cross-branch Windows

Duration (days)	0	59	98	120	196	never
Counts	15	2	1	1	1	6

(a) OpenSSL

Duration (days)	0	9	13	20	232	356	never
Counts	8	1	1	1	1	1	7

(b) GNU Crypto



(a) OpenSSL (b) GNU crypto

Fig. 3. Countermeasure types of the two libraries.

Response coverage. We found that *the majority of discovered vulnerabilities were addressed in OpenSSL and GNU Crypto, except that microarchitectural padding oracle vulnerabilities [167, 220] still exist in both libraries at the time of writing.* One possible reason is that such host attacks require stronger adversarial capabilities and can only work in limited contexts, and thus are less severe.

6.3 Cross-branch Patch Consistency

An application usually maintains different development branches concurrently. An adversary can still attack unpatched branches if the fix is not applied to all live branches at the same time. For instance, OpenSSL replaced the vulnerable sliding window scalar multiplication with branchless Montgomery ladder in version 1.1.0i on August 14, 2018, but not in the 1.0.2 branch. This left a chance for port-based attacks [7] to work on the sliding window implementation in OpenSSL 1.0.2, which urged the developers to apply the patch to 1.0.2q on November 20, 2018.

For each vulnerability, we measure the *cross-branch vulnerability window*, defined as the duration from the first patch release date to the date when all live branches are patched. Table 6 shows the number of patches in different vulnerability windows for both libraries. *In most cases, a patch was applied to all live branches at the same time (0 days). Some patches are, however, still missing in certain branches at the time of writing (never).* For example, OpenSSL 1.0.1 introduced masked-window multiplication and AES-NI support that were never ported to 0.9.8 and 1.0.0 branches before their end of life. OpenSSL 1.0.2r includes a bug fix for protocol error handling, but it is not applied to 1.1.0 and 1.1.1. Some new side-channel bug fixes—not critical, though—in OpenSSL 1.1.1 and 1.1.1b were not included in 1.0.2 and 1.1.0. For GNU Crypto, CVE-2015-0837 was fixed in GnuPG 1.4.19 and Libgcrypt 1.6.3, but not in Libgcrypt 1.5.x. Fortunately, this branch has reached its end of life on December 31, 2016.

6.4 Countermeasure Type

We study four categories of countermeasures commonly adopted by cryptographic libraries to fix side-channel vulnerabilities: (1) introducing brand new implementations; (2) selecting existing secure implementations; (3) fixing software bugs; (4) enhancing robustness of existing implementations. Countermeasure classification for OpenSSL and GNU Crypto is shown in Figure 3.

Table 7. Side-channel Vulnerabilities in Other Applications

ID	CVE Date	C.	Application	Operations	CVE	Patch date
1	2001/06/27	■	OpenSSH, AppGate, ssh-1	RSA-PAD	2001-0361	2001/01/29
2	2004/12/31	■	MatrixSSL	Modular Multiplication	2004-2682	2004/06/01
3	2009/08/31	■	XySSL	RSA-PAD	2008-7128	●
4	2010/09/22	□	Microsoft IIS	Padding oracle attack	2010-3332	2010/09/27
5	2010/10/20	■	Apache MyFaces	Padding oracle attack	2010-2057	2010/06/10
6	2010/10/20	■	Oracle Mojarra	Padding oracle attack	2010-4007	2010/06/10
7	2013/02/08	■	Rack	HMAC comparison	2013-0263	2013/02/07
8	2013/02/08	■	Mozilla NSS	MAC-CBC-PAD	2013-1620	2013/02/14
9	2013/02/08	■	wolfSSL CyaSSL	MAC-CBC-PAD	2013-1623	2013/02/05
10	2013/02/08	■	Bouncy Castle	MAC-CBC-PAD	2013-1624	2013/02/10
11	2013/02/08	□	Opera	MAC-CBC-PAD	2013-1618	2013/02/13
12	2013/06/21	□	IBM WebSphere Commerce	Padding oracle attack	2013-0523	○
13	2013/10/04	■	PolarSSL	RSA-CRT	2013-5915	2013/10/01
14	2013/11/17	■	OpenVPN	Padding oracle attack	2013-2061	2013/03/19
15	2014/08/16	□	IBM WebSphere DataPower	-	2014-0852	○
16	2014/12/09	□	F5 BIG-IP	MAC-CBC-PAD	2014-8730	○
17	2015/07/01	■	Libcrypt++	Rabin-Williams DSA	2015-2141	2015/11/20
18	2015/08/02	□	Siemens RuggedCom ROS	MAC-CBC-PAD	2015-5537	○
19	2015/11/08	□	IBM DataPower Gateways	Padding oracle attack	2015-7412	○
20	2016/04/07	■	Erlang/OTP	MAC-CBC-PAD	2015-2774	2015/03/26
21	2016/04/12	□	EMC RSA BSAFE	RSA-CRT	2016-0887	○
22	2016/04/21	□	CloudForms Mgmt. Engine	Padding oracle attack	2016-3702	●
23	2016/05/13	■	Botan	MAC-CBC-PAD	2015-7827	2015/10/26
24	2016/05/13	■	Botan	Modular inversion	2016-2849	2016/04/28
25	2016/07/26	■	Cavium SDK	RSA-CRT	2015-5738	○
26	2016/09/03	■	jose-php	HMAC comparison	2016-5429	2016/08/30
27	2016/09/08	□	HPE Integrated Lights-Out 3	Padding oracle attack	2016-4379	2016/08/30
28	2016/10/10	■	Intel IPP	RSA	2016-8100	○
29	2016/10/28	■	Botan	RSA-PAD	2016-8871	2016/10/26
30	2016/12/13	■	wolfSSL	AES T-table lookup	2016-7440	2016/09/26
31	2016/12/15	□	Open-Xchange OX Guard	Padding oracle attack	2016-4028	2016/04/21
32	2017/01/23	■	Malcolm Fell jwt	Hash comparison	2016-7037	2016/09/05
33	2017/02/03	□	EMC RSA BSAFE	Padding oracle attack	2016-8217	2017/01/20
34	2017/02/13	■	Crypto++		2016-3995	2016/09/11
35	2017/03/03	■	MatrixSSL	RSA-CRT	2016-6882	2016/11/25
36	2017/03/03	■	MatrixSSL	RSA-PAD	2016-6883	2016/04/18
37	2017/03/07	■	Intel QAT	RSA-CRT	2017-5681	○
38	2017/03/23	□	Cloudera Navigator	MAC-CBC-PAD	2015-4078	○
39	2017/04/10	■	Botan	MAC-CBC-PAD	2015-7824	2015/10/26
40	2017/04/14	■	Nettle	Modular exponentiation	2016-6489	2016/08/04
41	2017/06/30	□	OSCI-Transport	Padding oracle attack	2017-10668	●
42	2017/07/27	■	Apache HTTP	Padding oracle attack	2016-0736	2016/10/20
43	2017/08/02	□	Citrix NetScaler	MAC-CBC-PAD	2015-3642	○
44	2017/08/10	■	Apache CXF	MAC comparison	2017-3156	○
45	2017/08/20	■	Nimbus JOSE+JWT	Padding oracle attack	2017-12973	2017/06/02
46	2017/09/25	■	Botan	Modular exponentiation	2017-15533	2017/10/02
47	2017/11/17	□	F5 BIG-IP	RSA-PAD	2017-6168	○
48	2017/12/12	■	Erlang/OTP	RSA-PAD	2017-1000385	2017/11/23
49	2017/12/12	■	Bouncy Castle	RSA-PAD	2017-13098	2017/12/28
50	2017/12/12	■	wolfSSL	RSA-PAD	2017-13099	2017/10/31
51	2017/12/13	□	Citrix NetScaler	RSA-PAD	2017-17382	○
52	2017/12/13	□	Radware Alteon	RSA-PAD	2017-17427	○
53	2017/12/15	□	Cisco ASA	RSA-PAD	2017-12373	2018/01/05
54	2017/12/28	■	Intel IPP		2018-3691	2018/05/22
55	2018/01/02	■	Linaro OP-TEE	RSA Montgomery	2017-1000413	2017/07/07
56	2018/01/10	□	Palo Alto Networks PAN-OS	RSA-PAD	2017-17841	○
57	2018/02/05	□	Cavium Nitrox and TurboSSL	RSA-PAD	2017-17428	○
58	2018/02/07	□	IBM GSKit	Padding oracle attack	2018-1388	○
59	2018/02/26	□	Unisys ClearPath MCP	RSA-PAD	2018-5762	○
60	2018/05/17	□	Symantec SSL Visibility	RSA-PAD	2017-15533	2018/01/12
61	2018/05/17	□	Symantec IntelligenceCenter	RSA-PAD	2017-18268	○
62	2018/06/04	■	Bouncy Castle	DSA	2016-1000341	2016/12/23
63	2018/06/04	■	Bouncy Castle	Padding oracle attack	2016-1000345	2016/12/23
64	2018/06/14	■	Mozilla NSS	Padding oracle attack	2018-12404	2018/12/07
65	2018/06/14	■	LibreSSL	Modulo primitive	2018-12434	2018/06/13
66	2018/06/14	■	Botan	Modulo primitive	2018-12435	2018/07/02
67	2018/06/14	■	wolfssl	Modulo primitive	2018-12436	2018/05/27
68	2018/06/14	■	LibTomCrypt	Modulo primitive	2018-12437	●
69	2018/06/14	■	LibSunEC	Modulo primitive	2018-12438	●
70	2018/06/14	■	MatrixSSL	Modulo primitive	2018-12439	2018/09/13
71	2018/06/14	■	BoringSSL	Modulo primitive	2018-12440	●
72	2018/07/28	■	ARM mbed TLS	MAC-CBC-PAD	2018-0497	2018/07/24
73	2018/07/28	■	ARM mbed TLS	MAC-CBC-PAD	2018-0498	2018/07/24
74	2018/07/31	□	Huawei products	RSA-PAD	2017-17174	○
75	2018/08/15	□	Clavister cOS Core	RSA-PAD	2018-8753	○
76	2018/08/15	□	ZyXEL ZyWALL/USG	RSA-PAD	2018-9129	○
77	2018/08/21	□	Huawei products	RSA-PAD	2017-17305	○
78	2018/08/23	■	Cloud Foundry Bits Service		2018-15800	2018/12/05
79	2018/08/31	□	RSA BSAFE Edition Suite	RSA-PAD	2018-11057	○
80	2018/09/11	□	RSA BSAFE SSL-J	RSA-PAD	2018-11069	○
81	2018/09/11	□	RSA BSAFE Crypto-J	RSA-PAD	2018-11070	○
82	2018/09/12	□	Intel AMT	RSA-PAD	2018-3616	○
83	2018/09/21	■	Apache Mesos	HMAC comparison	2018-8023	2018/07/25
84	2018/12/03	■	nettle	RSA-PAD	2018-16869	●
85	2018/12/03	■	wolfSSL	RSA-PAD	2018-16870	2018/12/27
86	2019/01/03	□	RSA BSAFE Crypto-C		2019-3731	2019/09/11
87	2019/01/03	□	RSA BSAFE Crypto-C		2019-3732	2018/08/28
88	2019/01/03	□	RSA BSAFE Crypto-J		2019-3739	2019/08/11
89	2019/01/03	□	RSA BSAFE Crypto-J		2019-3740	2019/08/11
90	2019/02/22	□	Citrix NetScaler Gateway	Padding oracle attack	2019-6485	○
91	2019/03/01	■	hostapd, wpa_supplicant		2019-9494	2019/04/21
92	2019/03/01	■	hostapd, wpa_supplicant		2019-9495	2019/04/21
93	2019/03/08	■	Botan	Scalar multiplication	2018-20187	2018/10/01
94	2019/03/26	■	Apache Tapestry	HMAC comparison	2019-10071	2019/09/07
95	2019/04/03	■	elliptic	Scalar multiplication	2019-10764	●
96	2019/04/11	□	Intel PTT, TXE, SPS		2019-11090	●
97	2019/07/07	■	hostapd, wpa_supplicant		2019-13377	2019/08/07
98	2019/07/17	■	wolfSSL, wolfCrypt	Scalar multiplication	2019-13628	2019/07/22
99	2019/07/17	■	MatrixSSL	Scalar multiplication	2019-13629	●
100	2019/07/27	■	Crypto++	Scalar multiplication	2019-14318	2019/07/29
101	2019/08/27	□	Fortinet FortiOS		2019-15703	2019/12/19
102	2019/09/26	■	ARM mbed TLS & Crypto		2019-16910	2019/09/06
103	2019/10/21	■	ARM mbed TLS & Crypto		2019-18222	2020/02/21
104	2019/12/05	■	Jenkins	TCP secret comparison	2020-2101	2020/01/29
105	2019/12/05	■	Jenkins	HMAC comparison	2020-2102	2020/01/29
106	2019/12/24	■	wolfSSL	Modulo multiplication	2019-19960	2019/12/20
107	2019/12/24	■	wolfSSL	Modulo inversion	2019-19963	2019/12/20
108	2020/01/22	■	Parity libsecp256k1-rs	Scalar overflow check	2019-20399	2019/10/02
109	2020/03/24	■	ARM mbed TLS	Modular inversion	2020-10932	2020/4/14
110	2020/04/12	■	wolfSSL	Modulo multiplication	2020-11713	2020/04/22

(■: open-source, □: closed-source; ●: Whether this vulnerability is addressed is not revealed. ○: This vulnerability is addressed, but the date is not revealed.)

In the earlier days, the primary fix for side-channel vulnerabilities in OpenSSL was to introduce new implementations. *After many years' evolution, every cryptographic operation now has secure implementations, and brand new solutions become unnecessary.* Recent patches were often minor bug fixes. Besides, previously developers only patched the code upon revelation of new issues. Now they proactively fortify the implementation without the evidence of potential vulnerabilities. This definitely improves the security of the library against side-channel attacks.

GNU Crypto has fewer vulnerabilities and patches compared to OpenSSL and prefers to use traditional solutions for some common issues. For instance, to mitigate the vulnerability in sliding window scalar multiplication, OpenSSL adopted a new solution, masked-window multiplication, while Libgcrypt regressed to less efficient double-and-add-always. Besides, development of GNU Crypto is generally several years behind that of OpenSSL.

6.5 Comparisons with Other Libraries

Finally, we summarize side-channel CVEs in other cryptographic applications (Table 7).

Table 8. Severity Comparisons

		CVSS			Count
		Base	Exploit	Impact	
OpenSSL	Asymmetric	3.14	6.09	2.90	10
	Protocol	4.25	8.46	2.90	13
GNU Crypto	Asymmetric	2.58	4.88	2.90	9
	Protocol	2.70	3.90	2.90	6
Open-source	Asymmetric	3.89	7.10	3.27	31
	Protocol	4.25	7.86	3.30	24
	Other	3.69	6.89	3.13	15
Closed-source	Asymmetric	2.60	4.90	2.90	1
	Protocol	4.36	8.43	3.09	32
	Other	4.28	8.47	2.90	7

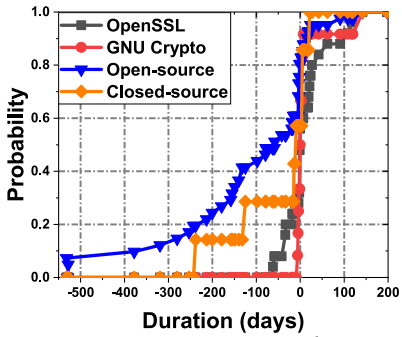


Fig. 4. Response speed.

Vulnerable Categories. Table 8 shows the breakdown of vulnerabilities in different categories. We observe that *vulnerabilities exist widely in many applications, besides OpenSSL and GNU Crypto.* We believe a lot of unrevealed vulnerabilities still exist in various applications, for two reasons.

First, researchers tend to study common cryptographic libraries, encouraging their developers to continuously improve the code. Other less-evaluated applications may still contain out-of-date vulnerabilities, but their developers are unaware or ignorant of them. For instance, RSA padding oracle attack was proposed 20 years ago and has been mitigated in common libraries such as OpenSSL and GnuTLS, but it still exists in about one-third of top 100 Internet domains, including Facebook and PayPal, as well as widely used products from IBM, Cisco, and so on [24].

Second, microarchitectural attacks usually require the source code to be available, prohibiting researchers from discovering vulnerabilities in closed-source applications. For instance, Table 8 shows that the majority of vulnerabilities found in closed-source applications are padding oracles via remote timing or message side channels, likely because no source code is needed to experiment with these attacks. We do not know if they also suffer from padding oracle attacks via microarchitectural side channels, as current studies [106, 166, 167, 220] evaluated them only on open-source libraries. It is also unclear if they possess vulnerabilities related to asymmetric ciphers for the similar reason.

Response speed and coverage. Figure 4 compares the response speeds of different applications. Interestingly, *they all responded to the vulnerabilities very fast.* Most vulnerabilities were published only after the release of corresponding patches, leaving no vulnerability windows to exploit.

Regarding the coverage, *most discovered vulnerabilities were addressed, with a few exceptions* (annotated with ● in Table 7) where too little public information is available. For these cases, we are unable to ascertain whether these issues were solved or not.

7 CONCLUSION

Microarchitectural side-channel attacks against cryptographic implementations have been an enduring topic over the past 20 years. Many vulnerabilities have been discovered from previous cryptographic implementations, but unknown ones likely still exist in today's implementations. The good news is that the community resolved these vulnerabilities very actively, and hence large-scale side-channel attacks causing severe real-world damages have not happened so far. Besides, years of efforts have fortified common cryptographic libraries and applications against side-channel attacks, and recently discovered vulnerabilities were less significant or surprising.

Looking ahead, we expect continuous arms race between side-channel attacks and defenses. We encourage researchers to discover new vulnerabilities and attacks, evaluate them on a wider range of applications, and develop novel countermeasures for them.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] Onur Aciiçmez. 2007. Yet another microarchitectural attack: Exploiting I-cache. In *ACM Workshop on Computer Security Architecture*.
[2] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. 2010. New results on instruction cache attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[3] Onur Acıiçmez, Shay Gueron, and Jean-Pierre Seifert. 2007. New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In *IMA International Conference on Cryptography and Coding*.
[4] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. 2007. Predicting secret keys via branch prediction. In *Cryptographers' Track at the RSA Conference*.
[5] Onur Acıiçmez, Werner Schindler, and Çetin K. Koç. 2007. Cache based remote timing attack on the AES. In *Cryptographers' Track at the RSA Conference*.
[6] Onur Aciicmez and Jean-Pierre Seifert. 2007. Cheap hardware parallelism implies cheap security. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*.
[7] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. 2019. Port contention for fun and profit. In *IEEE Symposium on Security and Privacy*.
[8] Alejandro Cabrera Aldaya, Cesar Pereida García, Luis Manuel Alvarez Tapia, and Billy Bob Brumley. 2019. Cache-timing attacks on RSA key generation. *IACR Trans. Cryptog. Hardw. Embed. Syst.* 2019, 4 (2019), 213–242.
[9] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, and Yuval Yarom. 2016. Amplifying side channels through performance degradation. In *Conference on Computer Security Applications*.
[10] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. 2016. Verifying constant-time implementations. In *USENIX Security Symposium*.
[11] Tiago Alves. 2004. Trustzone: Integrated hardware and software security. *White paper* (2004).
[12] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. 2015. On subnormal floating point and abnormal timing. In *IEEE Symposium on Security and Privacy*.
[13] Roberto Maria Avanzi. 2005. Side channel attacks on implementations of curve-based cryptographic primitives. *IACR Cryptology ePrint Archive*. 17.
[14] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. 2014. System-level non-interference for constant-time cryptography. In *ACM Conference on Computer and Communications Security*.
[15] Naomi Benger, Joop Van de Pol, Nigel P. Smart, and Yuval Yarom. 2014. "Ooh Aah . . . Just a Little Bit": A small amount of side channel can go a long way. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[16] Daniel J. Bernstein. 2005. Cache-timing attacks on AES. *Technical Report*. 3. Citeseer.
[17] Daniel J. Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruinderink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. 2017. Sliding right into disaster: Left-to-right sliding windows leak. In *International Conference on Cryptographic Hardware and Embedded Systems*.
[18] Johann Betz, Dirk Westhoff, and Günter Müller. 2017. Survey on covert channels in virtual machines and cloud computing. *Trans. Emerg. Telecommun. Technol.* 28, 6, (2017), e3134.
[19] Sarani Bhattacharya and Debdeep Mukhopadhyay. 2016. Curious case of Rowhammer: Flipping secret exponent bits using timing analysis. In *International Conference on Cryptographic Hardware and Embedded Systems*.
[20] Sarani Bhattacharya, Chester Rebeiro, and Debdeep Mukhopadhyay. 2012. Hardware prefetchers leak: A revisit of SVF for cache-timing attacks. In *IEEE/ACM International Symposium on Microarchitecture Workshops*.
[21] Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. 2017. A survey of timing channels and countermeasures. *ACM Comput. Surv.* 50, 1 (2017), 1–39.
[22] Sandrine Blazy, David Pichardie, and Alix Trieu. 2017. Verifying constant-time implementations by abstract interpretation. In *European Symposium on Research in Computer Security*.
[23] Daniel Bleichenbacher. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In *International Cryptology Conference*.
[24] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return of Bleichenbacher's oracle threat (ROBOT). In *Usenix Security Symposium*.
[25] Barry Bond, Chris Hawblitzel, Manos Kapritsos, K. Rustan M Leino, Jacob R. Lorch, Bryan Parno, Ashay Rane, Srinath Setty, and Laure Thompson. 2017. Vale: Verifying high-performance cryptographic assembly code. In *USENIX Security Symposium*.

[26] Joseph Bonneau and Ilya Mironov. 2006. Cache-collision timing attacks against AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[27] Jurjen Bos and Matthijs Coster. 1989. Addition chain heuristics. In *Conference on the Theory and Application of Cryptology*.
[28] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. In *USENIX Workshop on Offensive Technologies*.
[29] Benjamin A. Braun, Suman Jana, and Dan Boneh. 2015. Robust and efficient elimination of cache and timing side channels. *arXiv preprint arXiv:1506.00189* (2015).
[30] Ernie Brickell, Gary Graunke, Michael Neve, and Jean-Pierre Seifert. 2006. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. *IACR Cryptol. ePrint Arch.* (2006), 52.
[31] Ernie Brickell, Gary Graunke, and Jean-Pierre Seifert. 2006. Mitigating cache/timing based side-channels in AES and RSA software implementations. In *RSA Conference 2006 session DEV-203*.
[32] Samira Briongos, Pedro Malagón, Juan-Mariano de Goyeneche, and Jose M. Moya. 2019. Cache misses and the recovery of the full AES 256 key. *Appl. Sci.* 9, 5 (2019), 944.
[33] Samira Briongos, Pedro Malagón, José M. Moya, and Thomas Eisenbarth. 2020. RELOAD+ REFRESH: Abusing cache replacement policies to perform stealthy cache attacks. In *USENIX Security Symposium*.
[34] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. 2016. Flush, Gauss, and reload-A cache attack on the BLISS lattice-based signature scheme. In *International Conference on Cryptographic Hardware and Embedded Systems*.
[35] Billy Bob Brumley and Risto M. Hakala. 2009. Cache-timing template attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*.
[36] Billy Bob Brumley, Risto M. Hakala, Kaisa Nyberg, and Sampo Sovio. 2010. Consecutive S-box lookups: A timing attack on SNOW 3G. In *International Conference on Information and Communications Security*.
[37] Billy Bob Brumley and Nicola Tuveri. 2011. Remote timing attacks are still practical. In *European Symposium on Research in Computer Security*.
[38] David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. *Comput. Netw.* 48, 5 (2005), 701–716.
[39] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium*.
[40] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar et al. 2019. Fallout: Leaking data on meltdown-resistant cpus. In *ACM SIGSAC Conference on Computer and Communications Security*.
[41] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A systematic evaluation of transient execution attacks and defenses. In *USENIX Security Symposium*.
[42] Anne Canteaut, Cedric Lauradoux, and Andre Seznec. 2006. Understanding cache attacks. Doctoral dissertation, INRIA.
[43] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SGXPECTRE: Stealing Intel secrets from SGX enclaves via speculative execution. In *IEEE European Symposium on Security and Privacy*.
[44] Sanchuan Chen, Fangfei Liu, Zeyu Mi, Yinqian Zhang, Ruby B. Lee, Haibo Chen, and XiaoFeng Wang. 2018. Leveraging hardware transactional memory for cache side-channel defenses. In *ACM Asia Conference on Computer and Communications Security*.
[45] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *ACM Asia Conference on Computer and Communications Security*.
[46] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Appl. Soft Comput.* 49 (2016), 1162–1174.
[47] Jean-Sébastien Coron. 1999. Resistance against differential power analysis for elliptic curve cryptosystems. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[48] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.
[49] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*.
[50] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. 2015. Thwarting cache side-channel attacks through dynamic software diversity. In *Network and Distributed System Security Symposium*.
[51] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. 2018, 2 (2018), 171–191.

[52] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. Timing attacks on error correcting codes in post-quantum schemes. In *ACM Workshop on Theory of Implementation Security Workshop*.
[53] Mario Dehesa-Azuara, Matthew Fredrikson, Jan Hoffmann et al. 2017. Verifying and synthesizing constant-resource implementations with types. In *IEEE Symposium on Security and Privacy*.
[54] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: A metric for measuring information leakage. In *International Symposium on Computer Architecture*.
[55] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Comput. Archit. News* 41, 3 (2013), 559–570.
[56] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2018. Cache timing side-channel vulnerability checking with computation tree logic. In *International Workshop on Hardware and Architectural Support for Security and Privacy*.
[57] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2019. Analysis of secure caches using a three-step model for timing-based attacks. *J. Hardw. Syst. Secur.* 3, 4 (2019), 397–425.
[58] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2019. Secure TLBs. In *International Symposium on Computer Architecture*.
[59] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2020. HybCache: Hybrid side-channel-resilient caches for trusted execution environments. In *USENIX Security Symposium*.
[60] Jean-Francois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. 1998. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*.
[61] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. 2017. Prime+ abort: A timer-free high-precision l3 cache attack using intel TSX. In *USENIX Security Symposium*.
[62] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-monopolizable Caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.* 8, 4 (2012), 1–21.
[63] Goran Doychev and Boris Köpf. 2017. Rigorous analysis of software countermeasures against cache attacks. In *ACM SIGPLAN Notices*.
[64] Goran Doychev, Boris Köpf, Laurent Mauborgne, and Jan Reineke. 2015. Cacheaudit: A tool for the static analysis of cache side channels. *ACM Trans. Inf. Syst. Secur.* 18, 1 (2015).
[65] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice signatures and bimodal Gaussians. In *Cryptology Conference*.
[66] Nadia El Mrabet, Jacques J. A. Fournier, Louis Goubin, and Ronan Lashermes. 2015. A survey of fault attacks in pairing based cryptography. *Cryptog. Commun.* 7, 1 (2015), 185–205.
[67] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theor.* 31, 4 (1985), 469–472.
[68] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. 2017. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In *ACM Conference on Computer and Communications Security*.
[69] Dmitry Evtyushkin, Ryan Riley, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2018. BranchScope: A new side-channel attack on directional branch predictor. In *International Conference on Architectural Support for Programming Languages and Operating Systems*.
[70] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. 2010. State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures. In *IEEE International Symposium on Hardware-oriented Security and Trust*.
[71] Junfeng Fan and Ingrid Verbauwhede. 2012. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In *Cryptography and Security: From Theory to Applications*. Springer, Berlin, Heidelberg, 265–282.
[72] Shuqin Fan, Wenbo Wang, and Qingfeng Cheng. 2016. Attacking OpenSSL implementation of ECDSA with a few signatures. In *ACM Conference on Computer and Communications Security*.
[73] Andrew Ferraiuolo, Yao Wang, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. 2016. Lattice priority scheduling: Low-overhead timing-channel protection for a shared memory controller. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*.
[74] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the many sides of target row refresh. *arXiv preprint arXiv:2004.01807* (2020).
[75] Cesar Pereida García and Billy Bob Brumley. 2017. Constant-time callees with variable-time callers. In *USENIX Security Symposium*.
[76] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptog. Eng.* 8, 1 (2018), 1–27.

[77] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. 2018. Drive-by key-extraction cache attacks from portable code. In *International Conference on Applied Cryptography and Network Security*.
[78] Daniel Genkin, Itamar Pipman, and Eran Tromer. 2015. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. *J. Cryptog. Eng.* 5, 2 (2015), 95–112.
[79] Daniel Genkin, Adi Shamir, and Eran Tromer. 2014. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *International Cryptology Conference*.
[80] Daniel Genkin, Luke Valenta, and Yuval Yarom. 2017. May the fourth be with you: A microarchitectural side channel attack on several real-world applications of Curve25519. In *ACM Conference on Computer and Communications Security*.
[81] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *ACM symposium on Theory of Computing*.
[82] Benedikt Gierlichs, Lejla Batina, Christophe Clavier, Thomas Eisenbarth, Aline Gouget, Helena Handschuh, Timo Kasper, Kerstin Lemke-Rust, Stefan Mangard, Amir Moradi et al. 2008. Susceptibility of eSTREAM candidates towards side channel analysis. In *the State of the Art of Stream Ciphers, Workshop*. 123–150.
[83] Michael Godfrey and Mohammad Zulkernine. 2013. A server-side solution to cache-based side-channel attacks in the cloud. In *IEEE Sixth International Conference on Cloud Computing*.
[84] Daniel M. Gordon. 1998. A survey of fast exponentiation methods. *J. Algor.* 27, 1 (1998), 129–146.
[85] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *European Workshop on Systems Security*.
[86] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *USENIX Security Symposium*.
[87] Marc Green, Leandro Rodrigues-Lima, Andreas Zankl, Gorka Irazoqui, Johann Heyszl, and Thomas Eisenbarth. 2017. AutoLock: Why cache attacks on ARM are harder than you think. In *USENIX Security Symposium*.
[88] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *USENIX Security Symposium*.
[89] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: A fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*.
[90] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security Symposium*.
[91] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache games–Bringing access-based cache attacks on AES to practice. In *IEEE Symposium on Security and Privacy*.
[92] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-resolution side channels for untrusted operating systems. In *USENIX Annual Technical Conference*.
[93] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. 2005. Guide to elliptic curve cryptography. Springer Science & Business Media.
[94] Zecheng He and Ruby B. Lee. 2017. How secure is your cache against side-channel attacks? In *IEEE/ACM International Symposium on Microarchitecture*.
[95] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*.
[96] Xiaolu Hou, Jakub Breier, Fuyuan Zhang, and Yang Liu. 2019. Fully automated differential fault analysis on software implementations of block ciphers. *IACR Trans. Cryptog. Hardw. Embed. Syst* (2019), 1–29.
[97] Wei-Ming Hu. 1992. Reducing timing channels with fuzzy time. *J. Comput. Secur.* 1, 3–4 (1992), 233–254.
[98] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical timing side channel attacks against kernel space ASLR. In *IEEE Symposium on Security and Privacy*.
[99] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *International Symposium on High Performance Computer Architecture*.
[100] Muzammil Hussain, Ahmed Al-Haiqi, A. A. Zaidan, B. B. Zaidan, M. L. Mat Kiah, Nor Badrul Anuar, and Mohamed Abdulnabi. 2016. The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks. *Pervas. Mob. Comput.* 25 (2016), 1–25.
[101] Mehmet Sinan Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cache attacks enable bulk key recovery on the cloud. In *International Conference on Cryptographic Hardware and Embedded Systems*.
[102] Gorka Irazoqui, Kai Cong, Xiaofei Guo, Hareesh Khattri, Arun Kanuparthi, Thomas Eisenbarth, and Berk Sunar. 2017. Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries. *arXiv preprint arXiv:1709.01552* (2017).
[103] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. S $ A: A shared cache attack that works across cores and defies VM sandboxing—and its application to AES. In *IEEE Symposium on Security and Privacy*.

[104] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. MASCAT: Stopping microarchitectural attacks before execution. *IACR Cryptol. ePrint Arch.* 1196.
[105] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Wait a minute! A fast, Cross-VM attack on AES. In *International Workshop on Recent Advances in Intrusion Detection*.
[106] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2015. Lucky 13 strikes back. In *ACM Symposium on Information, Computer and Communications Security*.
[107] Saad Islam, Ahmad Moghimi, Ida Bruhns, Moritz Krebbel, Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. 2019. SPOILER: Speculative load hazards boost Rowhammer and cache attacks. In *USENIX Security Symposium*.
[108] Yeongjin Jang, Sangho Lee, and Taesoo Kim. 2016. Breaking kernel address space layout randomization with Intel TSX. In *ACM Conference on Computer and Communications Security*.
[109] Marc Joye and Sung-Ming Yen. 2002. The Montgomery powering ladder. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[110] Anatolii Alekseevich Karatsuba and Yu P. Ofman. 1962. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk* 145, 2 (1962), 293–294.
[111] Jonathan Katz, Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. 1996. *Handbook of Applied Cryptography*. CRC Press.
[112] Thierry Kaufmann, Hervé Pelletier, Serge Vaudenay, and Karine Villegas. 2016. When constant-time source yields variable-time binary: Exploiting curve25519-donna built with MSVC 2015. In *International Conference on Cryptology and Network Security*.
[113] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. 2016. A high-resolution side-channel attack on last-level cache. In *Design Automation Conference*.
[114] K. Keerthi, Indrani Roy, Chester Rebeiro, Aritra Hazra, and Swarup Bhunia. 2020. FEDS: Comprehensive fault attack exploitability detection for software implementations of block ciphers. *IACR Trans. Cryptog. Hardw. Embed. Syst* (2020), 272–299.
[115] Georgios Keramidas, Alexandros Antonopoulos, Dimitrios N. Serpanos, and Stefanos Kaxiras. 2008. Non deterministic caches: A simple and effective defense against side channel attacks. *Des. Autom. Embed. Syst.* 12, 3 (2008), 221–230.
[116] Jeremie S. Kim, Minesh Patel, A. Giray Yaglikci, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An experimental analysis of modern dram devices and mitigation techniques. *arXiv preprint arXiv:2005.13121* (2020).
[117] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. 2012. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud. In *USENIX Conference on Security Symposium*.
[118] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Comput. Archit. News* 42, 3 (2014), 361–372.
[119] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A defense against cache timing attacks in speculative execution processors. In *IEEE/ACM International Symposium on Microarchitecture*.
[120] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy*.
[121] Paul C. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *International Cryptology Conference*.
[122] Boris Köpf, Laurent Mauborgne, and Martín Ochoa. 2012. Automatic quantification of cache side-channels. In *International Conference on Computer-aided Verification*.
[123] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAMbleed: Reading bits in memory without accessing them. In *IEEE Symposium on Security and Privacy*.
[124] Adam Langley, Mike Hamburg, and Sean Turner. 2016. *Elliptic Curves for Security*. Technical Report.
[125] Gregor Leander, Erik Zenner, and Philip Hawkes. 2009. Cache timing analysis of LFSR-based stream ciphers. In *IMA International Conference on Cryptography and Coding*.
[126] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security Symposium*.
[127] Peng Li, Debin Gao, and Michael K. Reiter. 2014. StopWatch: A cloud architecture for timing channel mitigation. *ACM Trans. Inf. Syst. Secur.* 17, 2 (2014), 1–28.
[128] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache attacks on mobile devices. In *USENIX Security Symposium*.

[129] Moritz Lipp, Vedad Hažić, Michael Schwarz, Arthur Perais, Clémentine Maurice, and Daniel Gruss. 2020. Take a way: Exploring the security implications of AMD's cache way predictors. In *ACM Asia Conference on Computer and Communications Security*.
[130] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*.
[131] Fangfei Liu and Ruby B. Lee. 2014. Random fill cache architecture. In *IEEE/ACM International Symposium on Microarchitecture*.
[132] Fangfei Liu, Hao Wu, Kenneth Mai, and Ruby B. Lee. 2016. Newcache: Secure cache architecture thwarting cache side-channel attacks. *IEEE Micro* 36, 5 (2016), 8–16.
[133] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*.
[134] Xiaoxuan Lou, Fan Zhang, Zheng Leong Chua, Zhenkai Liang, Yueqiang Cheng, and Yajin Zhou. 2019. Understanding Rowhammer attacks through the lens of a unified reference framework. *arXiv preprint arXiv:1901.03538* (2019).
[135] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *International Conference on the Theory and Applications of Cryptographic Techniques*.
[136] Robert Martin, John Demme, and Simha Sethumadhavan. 2012. TimeWarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *International Symposium on Computer Architecture*.
[137] Victor S. Miller. 1985. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*.
[138] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*.
[139] Ahmad Moghimi, Jan Wichelmann, Thomas Eisenbarth, and Berk Sunar. 2019. Memjam: A false dependency attack against constant-time crypto implementations. *Int. J. Parallel Prog.* 47, 4 (2019), 538–570.
[140] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. 2020. TPM-FAIL:{TPM} meets timing and lattice attacks. In *USENIX Security Symposium*.
[141] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. 2020. CopyCat: Controlled instruction-level attacks on enclaves. In *29th {USENIX} Security Symposium ({USENIX} Security'20)*. 469–486.
[142] Bodo Möller. 2012. Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures. Retrieved from http://www.openssl.org/~bodo/tls-cbc.txt.
[143] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. 2005. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In *International Conference on Information Security and Cryptology*.
[144] John Monaco. 2018. SoK: Keylogging side channels. In *IEEE Symposium on Security and Privacy*.
[145] Peter L. Montgomery. 1987. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comput.* 48, 177 (1987), 243–264.
[146] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, and Guy Gogniat. 2018. Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters. In *International Workshop on Hardware and Architectural Support for Security and Privacy*.
[147] Onur Mutlu and Jeremie S. Kim. 2019. RowHammer: A retrospective. *IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst.* 39, 8 (2019), 1555–1571.
[148] Ani Nahapetian. 2016. Side-channel attacks on mobile and wearable systems. In *IEEE Consumer Communications & Networking Conference*. IEEE.
[149] Michael Neve and Jean-Pierre Seifert. 2006. Advances on access-driven cache attacks on AES. In *International Workshop on Selected Areas in Cryptography*.
[150] Michael Neve, Jean-Pierre Seifert, and Zhenghong Wang. 2006. Cache time-behavior analysis on AES. Technical Report, Princeton University.
[151] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: The case of AES. In *Cryptographers' Track at the RSA Conference*.
[152] Mairéad O'Hanlon and Anthony Tonge. 2005. *Investigation of Cache Timing Attacks on AES.*. https://www.computing.dcu.ie/wpapers/2005/0105.pdf.
[153] Mathias Payer. 2016. HexPADS: a platform to detect "stealth" attacks. In *International Symposium on Engineering Secure Software and Systems*.
[154] Chris Peikert. 2010. An efficient and parallel Gaussian sampler for lattices. In *Cryptology Conference*.
[155] Colin Percival. 2005. Cache missing for fun and profit. BSDCan.
[156] Cesar Pereida García, Billy Bob Brumley, and Yuval Yarom. 2016. Make sure DSA signing exponentiations really are constant-time. In *ACM Conference on Computer and Communications Security*.

[157] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. 2017. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In *ACM Conference on Computer and Communications Security*.
[158] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-CPU attacks. In *USENIX Security Symposium*.
[159] Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. 2020. Systematic analysis of randomization-based protected cache architectures. In *42nd IEEE Symposium on Security and Privacy*, Vol. 5. 2021.
[160] Moinuddin K. Qureshi. 2018. Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *IEEE/ACM International Symposium on Microarchitecture*.
[161] Moinuddin K. Qureshi. 2019. New attacks and defense for encrypted-address cache. In *ACM/IEEE International Symposium on Computer Architecture*.
[162] Chester Rebeiro, Debdeep Mukhopadhyay, Junko Takahashi, and Toshinori Fukunaga. 2009. Cache timing attacks on Clefia. In *International Conference on Cryptology in India*.
[163] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. 2017. Dude, is my code constant time? In *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*.
[164] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.
[165] Bruno Rodrigues, Fernando Magno Quintão Pereira, and Diego F. Aranha. 2016. Sparse representation of implicit flows with applications to side-channel detection. In *ACM International Conference on Compiler Construction*.
[166] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. 2019. The 9 lives of Bleichenbacher's cat: New cache attacks on TLS implementations. In *IEEE Symposium on Security and Privacy*.
[167] Eyal Ronen, Kenneth G. Paterson, and Adi Shamir. 2018. Pseudo constant time implementations of TLS are only pseudo secure. In *ACM Conference on Computer and Communications Security*.
[168] Indrani Roy, Chester Rebeiro, Aritra Hazra, and Swarup Bhunia. 2019. Safari: Automatic synthesis of fault-attack resistant block cipher implementations. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 39, 4 (2019), 752–765.
[169] Gururaj Saileshwar and Moinuddin K. Qureshi. 2019. Lookout for zombies: Mitigating flush+ reload attack on shared caches by monitoring invalidated lines. *arXiv preprint arXiv:1906.02362* (2019).
[170] Daniel Sanchez and Christos Kozyrakis. 2011. Vantage: Scalable and efficient fine-grain cache partitioning. In *International Symposium on Computer Architecture*.
[171] Sercan Sari, Onur Demir, and Gurhan Kucuk. 2019. FairSDP: Fair and secure dynamic cache partitioning. In *International Conference on Computer Science and Engineering*.
[172] Michael Schwarz and Daniel Gruss. 2020. How trusted execution environments fuel research on microarchitectural attacks. *IEEE Secur. Priv.* 18, 5 (2020), 18–27.
[173] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript template attacks: automatically inferring host information for targeted exploits. In *Network and Distributed System Security Symposium*.
[174] Michael Schwarz, Moritz Lipp, and Daniel Gruss. 2018. JavaScript zero: Real JavaScript and zero side-channel attacks. In *Network and Distributed System Security Symposium*.
[175] Michael Schwarz, Martin Schwarzl, Moritz Lipp, Jon Masters, and Daniel Gruss. 2019. NetSpectre: Read arbitrary memory over network. In *European Symposium on Research in Computer Security*.
[176] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*.
[177] Jicheng Shi, Xiang Song, Haibo Chen, and Binyu Zang. 2011. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*.
[178] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Network and Distributed System Security Symposium*.
[179] Youngjoo Shin, Hyung Chan Kim, Dokeun Kwon, Ji Hoon Jeong, and Junbeom Hur. 2018. Unveiling hardware-based data prefetcher, a hidden source of information leakage. In *ACM Conference on Computer and Communications Security*.
[180] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing page faults from telling your secrets. In *ACM on Asia Conference on Computer and Communications Security*.
[181] Joseph H. Silverman and William Whyte. 2007. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In *Cryptographers' Track at the RSA Conference*.
[182] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian. 2001. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security Symposium*.
[183] Read Sprabery, Konstantin Evchenko, Abhilash Raj, Rakesh B. Bobba, Sibin Mohan, and Roy Campbell. 2018. Scheduling, isolation, and cache allocation: A side-channel defense. In *IEEE International Conference on Cloud Engineering*.

[184] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard. 2018. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Commun. Surv. Tutor.* 20, 1 (2018), 465–488.
[185] Raphael Spreitzer, Gerald Palfinger, and Stefan Mangard. 2018. SCAnDroid: Automated side-channel analysis of android APIs. In *ACM Conference on Security & Privacy in Wireless and Mobile Networks*.
[186] Milind Srivastava, Patanjali Slpsk, Indrani Roy, Chester Rebeiro, Aritra Hazra, and Swarup Bhunia. 2020. SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware. In *Design, Automation & Test in Europe Conference & Exhibition*.
[187] Raoul Strackx and Frank Piessens. 2017. The Heisenberg defense: Proactively defending SGX enclaves against page-table-based side-channel attacks. *arXiv preprint arXiv:1712.08519* (2017).
[188] Jakub Szefer. 2018. Survey of microarchitectural side and covert channels, attacks, and defenses. *J. Hardw. Syst. Secur.* 3, 3 (2018), 219–234.
[189] Qinhan Tan, Zhihua Zeng, Kai Bu, and Kui Ren. 2020. PhantomCache: Obfuscating cache conflicts with localized randomization. In *Network and Distributed System Security Symposium*.
[190] Mehdi Tibouchi and Alexandre Wallet. 2019. *One Bit Is All It Takes: A Devastating Timing Attack on BLISS's Non-Constant Time Sign Flips*. Technical Report. Cryptology ePrint Archive, Report 2019/898.
[191] David Trilla, Carles Hernandez, Jaume Abella, and Francisco J. Cazorla. 2018. Cache side-channel attacks and time-predictability in high-performance critical real-time systems. In *Design Automation Conference*.
[192] Caroline Trippel, Daniel Lustig, and Margaret Martonosi. 2018. Checkmate: Automated synthesis of hardware exploits and security litmus tests. In *IEEE/ACM International Symposium on Microarchitecture*.
[193] Yukiyasu Tsunoo. 2002. Crypt-analysis of block ciphers implemented on computers with cache. *Proc. ISITA2002, Oct.* (2002).
[194] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. 2003. Cryptanalysis of DES implemented on computers with cache. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
[195] Michael Tunstall. 2017. Smart card security. In *Smart Cards, Tokens, Security and Applications*. Springer, Cham, 217–251.
[196] Johanna Ullrich, Tanja Zseby, Joachim Fabini, and Edgar Weippl. 2017. Network-based secret communication in clouds: A survey. *IEEE Commun. Surv. Tutor.* 19, 2 (2017), 1112–1144.
[197] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2017. SGX-Step: A practical attack framework for precise enclave execution control. In *Workshop on System Software for Trusted Execution*.
[198] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *USENIX Security Symposium*.
[199] Joop van de Pol, Nigel P. Smart, and Yuval Yarom. 2015. Just a little bit more. In *Cryptographers' Track at the RSA Conference*.
[200] Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift. 2014. Scheduler-based defenses against cross-VM side-channels. In *USENIX Security Symposium*.
[201] Bhanu C. Vattikonda, Sambit Das, and Hovav Shacham. 2011. Eliminating fine grained timers in Xen. In *ACM Workshop on Cloud Computing Security*.
[202] Serge Vaudenay. 2002. Security flaws induced by CBC padding-applications to SSL, IPSEC, WTLS .... In *International Conference on the Theory and Applications of Cryptographic Techniques*.
[203] Limin Wang, Ziyuan Zhu, Zhanpeng Wang, and Dan Meng. 2020. Analyzing the security of the cache side channel defences with attack graphs. In *Asia and South Pacific Design Automation Conference*.
[204] Ruisheng Wang and Lizhong Chen. 2014. Futility scaling: High-associativity cache partitioning. In *IEEE/ACM International Symposium on Microarchitecture*.
[205] Shuai Wang, Yuyan Bao, Xiao Liu, Pei Wang, Danfeng Zhang, and Dinghao Wu. 2019. Identifying cache-based side channels through secret-augmented abstract interpretation. In *USENIX Security Symposium*.
[206] Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu. 2017. Cached: Identifying cache-based timing channels in production software. In *USENIX Security Symposium*.
[207] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM SIGSAC Conference on Computer and Communications Security*.
[208] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. 2014. Timing channel protection for a shared memory controller. In *IEEE International Symposium on High Performance Computer Architecture*.
[209] Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, and G. Edward Suh. 2016. SecDCP: secure dynamic cache partitioning for efficient timing channel protection. In *Design Automation Conference*.
[210] Yao Wang and G. Edward Suh. 2012. Efficient timing channel protection for on-chip networks. In *IEEE/ACM Sixth International Symposium on Networks-on-Chip*.

[211] Yao Wang, Benjamin Wu, and G. Edward Suh. 2017. Secure dynamic memory scheduling against timing channel attacks. In *IEEE International Symposium on High Performance Computer Architecture*.
[212] Zhenghong Wang and Ruby B. Lee. 2006. Covert and side channels due to processor architecture. In *Computer Security Applications Conference*.
[213] Zhenghong Wang and Ruby B. Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *ACM International Symposium on Computer Architecture*.
[214] Zhenghong Wang and Ruby B. Lee. 2008. A novel cache architecture with enhanced performance and security. In *IEEE/ACM International Symposium on Microarchitecture*.
[215] Samuel Weiser, Raphael Spreitzer, and Lukas Bodner. 2018. Single trace attack against RSA key generation in Intel SGX SSL. In *Asia Conference on Computer and Communications Security*.
[216] Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl. 2018. DATA–differential address trace analysis: Finding address-based side-channels in binaries. In *USENIX Security Symposium*.
[217] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. 2019. ScatterCache: Thwarting cache attacks via cache set randomization. In *USENIX Security Symposium*.
[218] Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. 2018. MicroWalk: A framework for finding side channels in binaries. In *ACM Computer Security Applications Conference*.
[219] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. 2018. Eliminating timing side-channel leaks using program repair. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*.
[220] Yuan Xiao, Mengyuan Li, Sanchuan Chen, and Yinqian Zhang. 2017. STACCO: Differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves. In *ACM Conference on Computer and Communications Security*.
[221] Wenjie Xiong and Jakub Szefer. 2020. Leaking information through cache LRU states. In *IEEE International Symposium on High Performance Computer Architecture*.
[222] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *IEEE Symposium on Security and Privacy*.
[223] Mengjia Yan, Bhargava Gopireddy, Thomas Shull, and Josep Torrellas. 2017. Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks. In *ACM/IEEE International Symposium on Computer Architecture*.
[224] Yuval Yarom and Naomi Benger. 2014. Recovering OpenSSL ECDSA nonces using the FLUSH+ RELOAD cache side-channel attack. *IACR Cryptology ePrint Archive* 2014 (2014), 140.
[225] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*.
[226] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: a timing attack on OpenSSL constant-time RSA. *J. Cryptog. Eng.* 7, 2 (2017), 99–112.
[227] Sebastian Zander, Grenville Armitage, and Philip Branch. 2007. A survey of covert channels and countermeasures in computer network protocols. *IEEE Commun. Surv. Tutor.* 9, 3 (2007), 44–57.
[228] Andreas Zankl, Johann Heyszl, and Georg Sigl. 2016. Automated detection of instruction cache leaks in modular exponentiation software. In *International Conference on Smart Card Research and Advanced Applications*.
[229] Erik Zenner. 2008. A cache timing analysis of HC-256. In *International Workshop on Selected Areas in Cryptography*.
[230] Erik Zenner. 2009. Cache timing analysis of eStream finalists. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[231] Danfeng Zhang, Aslan Askarov, and Andrew C. Myers. 2012. Language-based control and mitigation of timing channels. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.
[232] Danfeng Zhang, Yao Wang, G. Edward Suh, and Andrew C. Myers. 2015. A hardware design language for timing-sensitive information-flow security. *ACM Sigplan Not.* 50, 4 (2015), 503–516.
[233] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. 2018. Persistent fault analysis on block ciphers. In *ACR Transactions on Cryptographic Hardware and Embedded Systems*. 150–172.
[234] Tianwei Zhang and Ruby B. Lee. 2014. New models of cache architectures characterizing information leakage from cache side channels. In *Computer Security Applications Conference*.
[235] Tianwei Zhang, Fangfei Liu, Si Chen, and Ruby B. Lee. 2013. Side channel vulnerability metrics: The promise and the pitfalls. In *International Workshop on Hardware and Architectural Support for Security and Privacy*.
[236] Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee. 2016. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*.
[237] Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee. 2018. Analyzing cache side channels using deep neural networks. In *Computer Security Applications Conference*.

[238] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *ACM Conference on Computer and Communications Security*.

[239] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2014. Cross-tenant side-channel attacks in PaaS clouds. In *ACM Conference on Computer and Communications Security*.

[240] Yinqian Zhang and Michael K. Reiter. 2013. DüPpel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *ACM Conference on Computer and Communications Security*.

[241] Xin-jie Zhao, Tao Wang, and Yuanyuan Zheng. 2009. Cache timing attacks on camellia block cipher. *IACR Cryptol. ePrint Arch.* (2009). 354.

[242] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. 2017. Camouflage: Memory traffic shaping to mitigate timing attacks. In *IEEE International Symposium on High Performance Computer Architecture*.

[243] Ziqiao Zhou, Michael K. Reiter, and Yinqian Zhang. 2016. A software approach to defeating side channels in last-level caches. In *ACM SIGSAC Conference on Computer and Communications Security*.

Received July 2020; revised January 2021; accepted March 2021