electronics

MDPI

Article

Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: a Survey of Potent Microarchitectural Attacks

Apostolos P. Fournaris [1,2,*], Lidia Pocero Fraile [1] and Odysseas Koufopavlou [1]

[1] Electrical and Computer Engineering Department University of Patras, Rion Campus, Patras 26500, Greece; lidiapf@upatras.gr (L.P.F.); odysseas@ece.upatras.gr (O.K.)

[2] Industrial Systems Institute, Research Center ATHENA, Platani, Patra 26504, Greece

* Correspondence: apofour@ece.upatras.gr; Tel.: +30-261-099-6822

Received: 31 May 2017; Accepted: 4 July 2017; Published: 13 July 2017

Abstract: Cyber-Physical system devices nowadays constitute a mixture of Information Technology (IT) and Operational Technology (OT) systems that are meant to operate harmonically under a security critical framework. As security IT countermeasures are gradually been installed in many embedded system nodes, thus securing them from many well-know cyber attacks there is a lurking danger that is still overlooked. Apart from the software vulnerabilities that typical malicious programs use, there are some very interesting hardware vulnerabilities that can be exploited in order to mount devastating software or hardware attacks (typically undetected by software countermeasures) capable of fully compromising any embedded system device. Real-time microarchitecture attacks such as the cache side-channel attacks are such case but also the newly discovered Rowhammer fault injection attack that can be mounted even remotely to gain full access to a device DRAM (Dynamic Random Access Memory). Under the light of the above dangers that are focused on the device hardware structure, in this paper, an overview of this attack field is provided including attacks, threat directives and countermeasures. The goal of this paper is not to exhaustively overview attacks and countermeasures but rather to survey the various, possible, existing attack directions and highlight the security risks that they can pose to security critical embedded systems as well as indicate their strength on compromising the Quality of Service (QoS) such systems are designed to provide.

Keywords: embedded system security; microarchitecture attacks; cache attacks; rowhammer attack

1. Introduction

Embedded systems are gradually gaining a considerable market and technology share of computational system devices in various domains. Some of those domains, like Critical Infrastructure (CI) environments, have a need for fast responsiveness thus requiring real-time embedded system but in parallel, they also have a high demand for strong security. Cybersecurity attackers can find fertile ground in the embedded system domain since such systems' original structure was not meant to provide security but rather high QoS and in several occasions real time response. Under this framework, actual QoS in a real-time embedded system can only be achieved when security is included as a QoS parameter.

A wide range of attacks from the cyber security domain can be mounted on real-time embedded system devices. Many of them are thwarted by IT based countermeasures including antimalware programs, firewalls, Intrusion Detection Systems and Anomaly detection tools. Recently, in the H2020 European CIPSEC (Enhancing Critical Infrastructure Protection with Innovative SECurity Framework) project, the above tools are considered as a unified whole, thus structuring a cyberattack protection framework that is applicable to many different Critical infrastructure environments that involve real-time embedded systems [1]. In this framework, apart from Critical Infrastructure, end devices mostly being embedded systems are also identified as a possible target of a security attacker. Thus, security strengthening of embedded system end devices is suggested through both hardware and software means.

Embedded systemend nodes, as most computing units, cannot be considered trusted due to many known vulnerabilities. Typically, software tools (e.g., antivirus, antimalware, firewalls) can protect a system from attackers taking advantage of those vulnerabilities to inject some malicious software. However, there exist security gaps and vulnerabilities that cannot be traced by the above-mentioned software tools since they are not stationed on software applications, drivers or the Operating System but rather on the computer architecture and hardware structure itself. Creating software to exploit these vulnerabilities remains hidden from most software cybersecurity tools and thus constitute a serious security risk for all devices using these commodity, vulnerable, computer structures.

One of the most potent means of enhancing computer security without compromises in QoS (i.e., real-time responsiveness, computation and network performance, power consumption etc.) is to migrate security functions in hardware. For this reason, there exist many hardware security elements along with appropriate software libraries that advertise security and trust. Trusted Platform Modules creating Trusted Execution Environment are very strong trust establishment solutions that provide secure, isolated from potential attacks, execution of sensitive applications [2–4]. This is achieved by creating isolated memory areas to execute code and store sensitive data, as proposed and implemented by ARM processor TrustZone technology [4]. These solutions, that are gradually been introduced in security critical areas (e.g., in CI systems), however, cannot be fully protected against hardware-based vulnerabilities.

Typical computer hardware components, like memory modules and caches, exhibit strange behavior under specific circumstances thus enabling backdoor access to potential attackers. Memory disturbance errors are observed in commodity DRAM chips stationed and used in all CI devices. It was pointed out that when a specific row of a DDR (Double Data Rate) memory bank is accessed repeatedly, i.e., is opened (i.e., activated) and closed (i.e., precharged) within a DRAM refresh interval, one or more bit flips occur in physically-adjacent DRAM rows to a wrong value. This DRAM disturbance error has been known as RowHammer [5].

Several researchers have observed that Rowhammer can be exploited to mount an attack and bypass most of the established software security and trust features including memory isolation by writing appropriate malicious software thus introducing memory disturbance attacks or RowHammer attacks. Such attacks can be used to corrupt system memory, crash a system, obtain and modify secret data or take over the entire system.

Furthermore, the security related computation on a real-time embedded system device can leak the sensitive information being processed to a knowledgeable attacker. These types of attacks, denoted as side channel attacks (SCAs) aim at collecting leakage data during processing and through statistical computations extract sensitive information. SCAs can be very potent to embedded system devices left unwatched for long periods of time so that an attacker can have physical access or proximity to them. Characteristic examples of such unattended devices are critical infrastructure systems' end nodes (e.g., sensors, in field actuators, PLCs (PLC: Programmable Logic Controller) or other monitoring/control devices). However, even if physical access to devices is not possible, there exist SCA types, like microarchitectural/cache SCAs, that can be mounted remotely.

Under the light of the above dangers, focused on a device's architecture and hardware structure, embedded system devices that are traditionally designed and built mainly for high QoS (reliability, high response time, high performance) and real-time responsiveness are left unprotected. In this paper, an overview of this attack research field is provided focusing attack directions and countermeasures. We provide a description and analysis of the attack possibilities and targets that exploit the rowhammer vulnerability as well as microarchitectural/cache SCAs that can be mounted from software and use computer architecture leakage channels for exploitation. While the above attacks can also be mounted remotely, in this paper, for the sake of completeness, a small overview is also provided for other SCAs that are very powerful when physical access to a device under attack is possible. All the attacks in this paper can bypass traditional cybersecurity software tools. The goal of this work is to help real time embedded system security experts/architects understand the above-mentioned attacks and motivate them to take appropriate countermeasures as those sketched in the paper.

The rest of the paper is organized as follows. In Section 2, the Rowhammer vulnerability is defined and its triggering mechanism is described. In Section 3, attacks using Rowhammer vulnerability are described for x86 and ARM embedded systems. In Section 4 side channel attacks that are applicable to embedded system devices are described focusing on microarchitecture attacks that can be mounted remotely. Finally, Section 5 provides conclusions and future research directions in this research field.

2. The Rowhammer Vulnerability

In the quest to get memories smaller, to reduce the cost-per-bit, and make them faster, vendors have reduced the physical geometry of DRAMs and increased density on the chip. But smaller cells can hold a smaller, limited amount of charge which reduces its noise margin and renders it more vulnerable to data loss. Also, the higher cell's proximity introduces electromagnetic coupling effects between them. And the higher variation in process technology increases the number of cells susceptibility to inter-cell crosstalk. Therefore, new DRAM technologies are more likely to suffer from disturbance that can go beyond their merges and provoke errors.

The existence and widespread nature of disturbance errors in the actual DRAM chips were exposed firstly in [5] at 2014. In this work, the root of the problem was identified in the voltage fluctuation on internal wires called wordline. Each row has its own *wordline* whose voltage is rising on a row access. Then, many accesses to the same row force this line to be toggled on and off repeatedly provoking voltage fluctuation that induce a disturbance effect on nearby rows. The perturbed rows leak charge at accelerated rate and if its data is not restored fast enough some of the cells changes its original value. In fact, it has been found that this vulnerability exists in the majority of the recent commodity DRAM chips, being more prevalent on 40 nm memory technologies. In the work of Carnegie Mellon University [5] this error phenomenon was found to exist in 139 DDR3 DRAM modules on x86 processors, for all the manufacturer modules from 2013 to 2014. Lanteigne performed an analysis on DDR4 memory [6] proving that rowhammer are certainly reproducible on such technology too, in March 2016. From the 12 modules under test, 8 of them show bit flips during theirs experiments under default refresh rate. And most recently in the paper [7], where a determinist Rowhammer Attack has been proposed for Mobile Platforms, it is observed that the majority of LPDDR3-devices (Low Power DDR memory devices) under test have induced flips and that even LPDDR2 is vulnerable, under ARMv7 based processors. Also, DDR4 memory modules, that include several countermeasures, remain vulnerable to the Rowhammer bug, as described in [8] where a Rowhammer based attack on DDR4 vulnerability is proposed.

The reasons behind the rowhammer vulnerability bug can be traced back to the diverse ways of interaction that provoke this specific disturbance as hypothesized in [5]. Note that this is not based on DRAM chip analysis but rather on past studies. Changing the voltage of a wordline could inject noise into an adjacency wordline through electromagnetic coupling and partially enabling it can cause leakage on its row's cells.

Toggling the wordline could accelerate the flow of charge between two bridge cells. Bridges are a class of DRAM faults in which conductive channels are formed between unrelated wires or capacitors. The hot-carrier injection effect can permanently damage a wordline and can be produced by toggling for 100 hours the same wordline as proven in [5]. Then, some of the hot-carrier can be injected on a neighboring row and modify their cells charge or alter the characteristics of their access transistors.

Triggering Rowhammer

By repeatedly accessing, hammering, the same memory row (aggressor row) an attacker can cause enough disturbance in a neighboring row (victim row) to cause a bit flip. This can be software triggering by code with a loop that generates millions of reads to two different DRAM rows of the same bank in each iteration.

The ability to make this row activation fast enough is the first prerequisite to trigger the bug. The Memory Controller must allow this fast access but generally, the biggest challenge is overpassing the several layers of cache that mask out all the CPU (Central Processing Unit) memory reads.

A memory access consists of stages. In ACTIVE stage, firstly a row is activated so as to transfer the data row to the bank's row buffer by toggling ON its specific associated wordline. Secondly, the specific column from the row is read/written (READ/WRITE stage) from or to the row buffer. Finally, the row is closed, by precharging (PRECHARGE stage) the specific bank, writing back the value to the row and plugging OFF the wordline. The disturbance error is produced on a DRAM row when a nearby wordline voltage is toggled repeatedly, meaning that it is produced on the repeated ACTIVE/PRECHARGE of rows and not on the column READ/WRITE stage. When a software code is triggering the Rowhammer bug by repeated accesses to the same physical DRAM address, care must be taken in order to ensure that each access will correspond to a new row activation. If the same physical address is accessed continuously, the corresponding data is already in the row buffer and no new activation is produced. Therefore, we must access two physical addresses that correspond to rows on the same bank to be sure that the *row buffer* is cleaning between memory access.

A second challenge to consider is the need of finding physical addresses that are mappings on rows from the same bank (single-side rowhammer*).* Early projects confronted this challenges by picked random addresses following probabilistic approaches [5,9–11]. In latest works, the exact physical address mapping of all the banks should be known in order to access both rows that are directly above and below the *victim* one to mount a *double-side rowhammer* attack. This attack is much more efficient and can guarantee a deterministic flip of a bit, on vulnerable DRAM memory, for specific chosen victim address location [7,8,12].

The capability to induce disturbance errors on specific systems depends on the toggle rate in relation with the refresh interval. This refers to how much times a specific row is activated between *refresh* commands that recharge the data on each cell. Generally, the more times the row is open on a refresh interval the bigger is the probability of bit flips. Nevertheless, when the memory access rate overpasses a limit, smaller number of bits are being flip than the typical case [7]. The memory controller could schedule refreshes and memory access commands under pressing memory accessing conditions [5]. Then, No Operation Instructions (NOP) on each integration of the loop can help sometimes to trigger more flip bits in order to lower the access memory rate to the same address [7]. On other hand, at a sufficiently high refresh interval, the errors could be completely eliminated but with a corresponding consequence on both power consumptions and performance [5,7,13].

Depending on the DRAM implementation and due to the intrinsic orientation property, some cells (true-cells) represent a logical value of '1' using the charge state while others cell (anti-cells) represent a logical value of '1' using the discharge state. There are cases where both orientations are used in distinct parts of the same DRAM module. For triggering the flip on bits, the cells must be charged to increase its discharge rate in order to trigger rowhammer. This means that only logical value '1' can produce flips for *true-cells* while logical value '0' data can produce flips for *anti-cells.* Therefore, the probability of flip bits on a specific row depends on the data that are kept in this row as well as the orientation of the specific row cells [5]. If the orientation of the modules is not known before the attack then a different data pattern test must be realized in order to check for the rowhammer vulnerability on specific system [5,6,14].

3. Rowhammer Attack

Hardware fault-attacks typically need to expose the device to specific physical conditions outside of its specification which require physical access to the device under attack. However, through rowhammer attack hardware faults are induced by software without a need to have direct physical access to the system.

In this section, the challenges of triggering rowhammer in a security-relevant manner will be described by summarizing existing exploitation techniques under different architectures, OS (Operating System) and already deployment countermeasures. To perform a successful end-to-end rowhammer attack the following four steps must be implemented/followed.

3.1. Determine the Device under Attack Specific Memory Architecture Characteristics

For probabilistic rowhammer induction approaches, it is enough to know the *row size* so that addresses on the same banck can be found [10]. But the physical address of the bits that correspond to each row, bank, DIMMs (Dual In-line Memory Module) and Memory channel is crucial to perform deterministic *double-side rowhammer*. This DRAM address mapping schemes used by the CPUS's memory controllers is not publicly known by major chip companies like Intel and ARM. Reverse engineering of the *physical address mapping* is proposed for Intel Sandy Bridge CPUs in [15] and used for the *double-side rowhammer* approach in [14]. The DRAMA paper [8] presented two other approaches in order to solve the issue: the first one uses physical probing of the memory bus through a high-bandwidth oscilloscope by measuring the voltage on the pins at the DIMM slots, the second uses *time analysis* that is based on the *rowbuffer conflict* so that it can find out address pair belonging to the same bank and then use this address set to reconstruct the specific map function automatically. Similarly, the authors in [16] are presenting their own specific algorithm based too on timing analysis based approach. The DRAMA solution with exhaustive search avoids the need to know the complex logic behind the memory addressing with a cost that is smaller than the algorithm proposed in [16]. Note that both approaches can be used on Intel or ARM platforms.

3.2. Managing to Activate Rows in Each Bank Fast Enough to Trigger the Rowhammer Vulnerability

Various methods have been used to bypass, clean or nullify all levels of cache in order to have direct access to the physical memory. The eviction of cache lines corresponding to the aggressor row addresses between its memory accesses has been approached in many ways. The first research approaches use directly the CFLUSH command available on user space (userland) for x86 devices [5,10,14]. Early software level countermeasures regarding the above approach were described in [10]. However, such countermeasures were overpassed and new attacks were implemented on different attack interfaces such as scripting language based attacks using web browsers that are triggered remotely [17]. In line with this research direction, new, different, eviction strategies have been proposed as a replacement for the flush instructions. The aim of such strategies is to find an eviction set that contains addresses belonging to the same cache set of the aggressor rows. Such addresses are accessed one after the other so as to force the eviction of our aggressor row. This can be achieved through the use of a timing attack to find out the eviction set [12,17,18], or by using reverse engineering study of the Device under attack, focusing on manipulating the complex hash functions used by modern Intel processor to further partition the cache into slices [9]. On another approach proposed in [1], rowhammer is triggered with non-temporal store instructions, specifically *libc* functions, taking advantages of its cache bypass characteristic. In the most recent studies under ARM architecture, the use of the Direct Memory Access memory management mechanism is proposed as the best way of bypassing CPUs and their caches [7].

3.3. Access the Aggressor Physical Address from Userland

In many cases access mechanisms, memory physical addresses from OS userland is a non-trivial action because generally, the unprivileged processes use virtual memory address space and the virtual to physical map interface is not public knowledge or has been no longer allowed in many OS versions after 2015 (acting as a rowhammer countermeasure) [19]. Attacks that was mounted after consulting their own pagemap interface to obtain the corresponding physical address of each element have been relegated to be applicable just on Linux kernel versions at most 4.0 [12,18]. Exploitation technics based on probabilistic approaches that just pick a random virtual address to trigger the bug [10,11] are based on test code [14] and on the native code of the work in [17]. On the other hand, physical addresses can be accessed through the use of huge virtual pages that are backed by contiguous physical addresses which can use relative offsets to access specific physical memory pages [20]. Note that for the rowhammer bug triggered from a scripting language (e.g., JavaScript) large type arrays that are allocated on 2MB regions are being used and a tool that translates JavaScript arrays indices to physical addresses is required to trigger the bit flips on known physical memory locations [9,17].

3.4. Exploit the Rowhammer Vulnerability (Take Advantage of the Bit Flips)

In the attack of [10], a probabilistic exploitation strategy that sprays the memory with page tables is proposed, hoping that at least one of them lands on a vulnerable region. A second probabilistic exploitation uses a timing-based technique to learn in which bank secret data are kept in order to flip their bits and then just trigger the bug on the specific bank and hope to produce a flip bit of the row's secret data [18]. Other attacks rely on special memory management features such as memory deduplication [20], where the OS can be tricked to map a victim owner memory page on the attacker-controlled physical memory pages, or MMU (Memory Management Unit) Paravirtualization [16], where a malicious VM is allowed to trick a VM (Virtual Machine) monitor into mapping a page table into a vulnerable location. Also, specific target secret can be found and the victim component can be tricked to place it on a vulnerable memory region on deterministic approaches. Finally, a technique is proposed in [7] that is based on predictable memory reuse patterns of standard physical memory allocator.

3.5. Attacks on x86 Devices

Various exploitation techniques have been implemented that exploit the rowhammer vulnerability aiming at bypassing all current defenses and completely subvert the x86 systems.

The x86 instruction set architecture has been implemented in processors from Intel, Cirix, AMD, VIA and many other companies and is used in several new generations of embedded systems. Rowhammer attacks against such devices aiming directly in memory accesses and not the OS itself are hard to detect with traditional software tools like antimalware or Firewall products. The potential diversity of the attack can lead to a broad range of possible compromises including unauthorized accesses, denial of services, or cryptography key theft. Rowhammer attacks can also be used as a backdoor (an attack vector) for more complex attacks that would otherwise be detected and failed in x86 architectures.

Several Rowhammer attacks exploiting x86 architecture causing serious security breaches were described in [12]. In one such attack, by triggering Rowhammer vulnerability bit flips are caused in Native Client (NaCl) program thus achieving privileges escalation, escaping from the x86 NaCl and acquiring the ability to make OS system calls (syscalls).

In a different attack [10], physical memory is filled with page tables for a single process (nmap() function is used to repeatedly fill a file's data in memory). This mapping sprays the memory with page table entries (PTEs), that are used to translate the newly mmap()'ed virtual addresses [12]. Performing rowhammer in memory containing page tables, there is a non-trivial probability that some PTE through bit flipping will be changed so that it points to a physical page that contains a page table. This will give the application we are using access to its own page tables thus giving to the application user (i.e., the attacker) full Read/Write permission to a page table entry, an action that results eventually to access the full physical memory [10,12].

Rowhammer presents a purely software driven way of inducing fault injection attacks [21] for finding private keys by flipping bits in memory. Such an attack is mounted for RSA (Rabin Shamir Adleman) exponentiation where a bit flip is injected through rowhammer in the secret exponent [18] This presents a new threat to the security of the embedded systems because it unlocks a hardware attack type (fault injection attacks) that traditionally need physical access to devices. Based on the theory that a single fault signature is enough to leak the secret in an unprotected implementation, the attacker aims at bit flips on the exponent, however having user-level privileges on the system, he does not know where the secret is placed in LLC (Last Level Cache) and DRAM. Thus, the attacker needs to identify the bank in which the secret exponent resides. This goal in [18], is achieved through a spy process which uses timing analysis to identify the channel, bank and rank where the secret is mapped by probing on the execution time of the RSA decryption algorithm. Then, the *eviction set* that corresponds to the secret set must be found so as to bypass cache without flush instructions. Finally, the Rowhammer vulnerability can be triggered on the bunk, where the public key is stored, thus resulting, with high probability of a bit flip of this key. Then performing a fault injection analysis will reveal from the fault public key, the secret exponent key.

The exploit of cross-VM rowhammer attacks was proven to be successful in server-side machines by achieved private key exfiltration from HTTPS web server and code injection that can bypass password authentication on an OpenSSH (SSH: Secure Shell) server [16]. This lead to a serious threat on multi-tenant infrastructure clouds since user clouds may collocate their virtual machines on the same physical server sharing hardware resources, including DRAMS. The attacks use paravirtualization VM to break VM isolation and compromise the integrity and confidentially of co-located VMs. The vulnerability relies on the fact that the mapping between the pseudo-physical memory page to machine memory for the user VM space can be tricked to point to memory pages of other VM or even the hypervisor. The presented technique, called *page table replacement attack*, conducts rowhammer attacks to flip bits deterministically in a page directory entry (PDE), thus making in a pointer to a different page table.

Abusing the Linux's memory deduplication system (KSM) and using the rowhammer vulnerability, an attacker can reliably flip bits in any physical page of the software stack [20]. A VM can abuse memory deduplication to place on a control target physical page a co-hosted victim VM page and then exploit the rowhammer to flip a particular bit in the victim page without permission to write on that bit. Due to the high popularity of KSM in production clouds, this technique can have devastating consequents in common cloud settings.

A remote software-induced rowhammer fault injection has been implemented on script languages [17]. The attack is executed in sandbox JavaScript, that is enabled by default on every modern browser, therefore can be launched from any website. The specific implementation is independent of the instruction set of the CPU and its main challenge relies on performing an optimal cache eviction strategy as a replacement for the flush instruction. Another two different thread models are proposen [11] based on x86 non-temporal store method. They are very convenient because they can be employed to implement attacks with the widely used memset and memcpy functions from libc library, thus giving to almost every code the potential of rowhammering. One of the models can be used to implement a code from untrusted source and scape NaCl sandbox. The second model can be used to trigger rowhammer with existing benign code that does not compromise the security, for example, multimedia players or PDF readers by compromising inputs.

Both, script and native, language-based approaches mentioned above, achieve to trigger rowhammer on remote systems and constitute a basis for adapting the existing vulnerability exploitation techniques to gain root privileges with a remote attack and access all the physical memory of a target system. Both can beat the basic rowhammer countermeasure since they bypass the flush instruction (banned in some latest OS versions). The second approach follows a very simple attack methodology, while the JavaScript approach is the easier to be executed by a remote attacker.

3.6. Attack on ARM Architecture

Deterministic rowhammer attacks are feasible on commodity mobile platforms by using the end-to-end DRAMMER attack on Android [7]. In this work, the Android/ARM exploitation that was employed in order to mount Rowhammer attack was Google's ION memory management tool that provides DMA buffer management API's (Application Programming Interface) from user space. Modern computing platforms need to support efficient memory sharing between their several different hardware components as well as between devices and userland services. Thus, the OS must provide allocators that support operations to physical contiguous memory pages since most devices need this kind of DMA (Direct Memory Access) performance. The DMA buffer management API provides from userland uncached memory access to physical memory addressing directly to the DRAM, solving two of the more complex Rowhammer attack requirements (steps 1 and 3).

More specifically, in the DRAMMER attack approach, initially predictable reuse patterns of standard physical memory allocator are used in order to place the target security-sensitive data at a vulnerable position in the physical memory [20]. Then, a memory template is created by triggering rowhammer on a big part of the physical memory, looking for bit flips. Afterward, the vulnerable location where the secret is placed is chosen and then *memory massaging* is performed by exhausting available memory chunks of varying sizes so as to drive the physical memory allocator into a state that brings the device under attack in the specific vulnerable memory region that we have predicted.

The goal of the DRAMMER ARM-based attack in [7] is to root the android device. This is achieved by overwriting the control page table to probe kernel memory from user *struct cred* structures [7]. The empirical studies provided in [7] points out that this end-to-end attack takes from 30 s to 15 min on the most vulnerable phone. However, the most important conclusion from the attack is that it exposes the vulnerability of many of the tested ARMv7 devices to the rowhammer bug which was considered not possible till now thus opens the road for rowhammering a wide variety of ARM-based embedded systems that have DRAM.

This final conclusion has several repercussions on the security of the latest IT enabled embedded system devices that still have small memory capabilities. This also includes mobile phones that in some cases can be used for fast incident response (e.g., in crisis management or critical infrastructure incidents). As described in the previous paragraphs, mobile devices can be vulnerable to rowhammer based attacks so they constitute an attack entry point to the security critical system.

Apart from this, embedded system device can also be rowhammer attacked. Most of them do not rely on ARM processors but this is gradually changing as ARM is dominating the marker and provides security benefits like ARM Trustzone. Also, most of them, currently, rely on SRAM which is hard to rowhammer, but there also exist embedded system devices with high processing power like the Raspberry Pi that relies on LPDDR3 (Low Power DRAM has also been found to be susceptible to the rowhammer bug) and constitutes an interesting rowhammer attack target. Furthermore, there is the embedded system approach from Intel (Intel Edisson, Intel Galileo) that has very attractive performance capabilities for embedded system world applications and Internet of Things (IoT). Note that Intel Edisson or Galileo rely on the x86 instruction set Architecture and thus they are easily rowhammmer attacked especially since they also have LPDDR3.

As the above embedded solutions are bound to be integrated also in the new real-time system end nodes, like those of a critical infrastructure (e.g., a power plant, an airport, a bridge) the rowhammer attack vulnerability should be taken into account in choosing the appropriate device for deployment in the field.

4. Side-Channel Analysis Attacks in Embedded System Devices

Side-channel analysis attacks exploit a device under attack hardware characteristics leakage (power dissipation, computation time, electromagnetic emission etc.) to extract information about the processed data and use them to deduce sensitive information (cryptographic keys, messages etc.). An attacker does not tamper with the device under attack in any way and needs only make appropriate observations to mount a successful attack. Such observation can be done remotely or physically through appropriate tools. Depending on the observed leakage, the most widely used SCAs are microarchitectural/cache, timing, power dissipation, electromagnetic Emission attacks.

4.1. Microarchitectural/Cache Attacks

Microarchitectural side channel attacks exploit the operation performed in a computer architecture during software execution. Fundamentally, a computer architecture design aims at optimizing processing speed, thus computer architects have designed machines where processed data are strongly correlated with memory access and execution times. This data and code correlation acts as an exploitable side channel for Microarchitectural SCAs. Since cache is strongly related to execution/processing time, has high granularity and is lacking any access restrictions, it constitutes an ideal focal point for Microarchitectural SCAs. Microarchitectural attacks are impervious to access boundaries established at the software level, so they can bypass many cyberattack application layer software countermeasure tools and have managed to be effective even against restrictive execution environments like Virtual Machines (VMs) [22] or ARM TrustZone [23].

As in all SCAs, cache-based attacks aim at recovering secret keys when executing a known cryptographic algorithm. Typically, there exist access based cache attacks where an attacker extracts results from observing his own cache channel (measuring cache access time) and timing based cache attacks where the attacker extracts results from measuring a victim's cache hits and misses to infer access time [24]. In [24,25], the authors provide a more detailed categorization of four attack types. Type I attacks on Cache Misses due to external interference, Type II attacks on Cache Hits due to external interference kind of attacks, Type III attacks on Cache Misses due to internal interference, and Type IV attacks on Cache Hits due to internal interference. Interference can be considered external when it is present between the attacker's program and the victim's program while internal is the interference that is related only with the victim's program. Most Type I and II attacks are access based attacks while most Type II and IV attacks are timing based attacks.

One of the first cache attacks was proposed and implemented by Bernstein [26] on AES (Advanced Encryption Standard) algorithm where microarchitectural timing differences when accessing various positions of a loaded in memory AES SBox look up table, were exploited. At the same year, another access based attack from Percival was also proposed in [27] focusing on openSSL key theft and RSA. Later, the PRIME + PROBE and EVICT + TIME attack approaches were introduced in [28]. Both attacks relied on the knowledge of a cache state before an encryption and the capturing of the cache changes during encryption to deduce sensitive data. Cache attacks were also expanded so as to compromise public key cryptography algorithms like RSA as proposed in [27] and later in [29], demonstrating that such attacks are possible on the full spectrum of popular cryptography algorithms. Attacks became more potent after the proposal of the FLUSH + RELOAD attack, described in [30,31] which exploits the shared memory pages of OS libraries stored in the Last Level Cache (LLC) of any computer and similarly to sophisticated variations of the PRIME+PROBE attack [32] also focused on LLC, became applicable in cross core applications even against VM devices [22,32,33]. Furthermore, variations of the FLASH + RELOAD attack have been proposed for ARM-based systems thus providing strong implications of cache SCA vulnerabilities in ARM embedded systems (including embedded system nodes or Android-based mobile devices systems and ARM TrustZone Enabled processes) [23,34,35].

Through the relevant research history, microarchitecture SCAs have exploited several different covert channels existing in typical computer systems. L1 Cache is one of the main targets of microarchitecture/cache SCAs either holding data (data L1 cache) or instructions (instruction L1 cache). The small size of this cache type, as well as the separation between data and instructions, can help an attacker to monitor the entire cache and relate the collected leakage to specific data or instruction blocks. On the other hand, the fact that L1 (and L2) cache in modern multicore processor systems is core-private, makes SCAs related to L1 very difficult in such systems.

Another convert channel for microarchitecture/cache SCAs is related to the Branch Processing Unit (BPU). Observing if there are branch mispredictions (through time differences between correct and wrong predictions) can be also a way of inducing an attack. However, the BPU is not easily accessible to modern processors and the above mentioned time differences are so small that is hardly observable. Also, BPU is core-private thus fairs the same problems as L1 cache channel [22].

The LLC constitute the most advantageous channel for mounting microarchitecture/cache SCAs because it is shared between processor cores and is associated with the processor main memory in a way that enables the discrimination of LLC accesses from memory accesses with small error rate (due to significant time differences) [22,30].

4.2. Other SCAs: Power Analysis/Electromagnetic Emission Attacks

Apart from the above microarchitectural/cache SCAs for the shake of completeness it should also be mentioned that there exist a very broad variety of other SCAs that require physical contact or close proximity to a device under attack in order to be mounted successfully. This scenario is not unrealistic in the case of embedded systems since there exist many in field deployed such devices that can be physically accessed by an attacker without detection (in industrial areas, in critical infrastructures systems). Such SCAs are very potent and can be successful even against dedicated hardware security modules if the latter do not have some relevant installed countermeasure. Typically, the goal of the attacks is to retrieve cryptographic keys that are processed in a target device under attack from the leakage trace that is emanated during this execution. Apart from timing based SCAs that were already discussed in the previous subsections, the most popular SCAs are the ones exploiting the power dissipation of a target device, or the Electromagnetic emission (EM) that is leaking from the target device. Both attacks are of similar nature since their leakage is related to current fluctuations happening during a cryptographic operation execution.

To model SCAs we can adopt the approach described in [36,37]. Assume that we have a computation C that consists of series of O_0 or O_1 operations that require inputs X_0 and X_1 respectively (thus O_i (X_i) for i in {0, 1}). During processing of the C computation, each operation can be linked to an information leakage variable L_i. A side channel analysis attack is possible if there is some secret information s that is shared between O_i and its leakage L_i. The ultimate goal of a side channel analysis is, by using an analysis strategy, to deduce s (secret value) from the information leakage L_i. The simplest way to achieve that is by examining a sequence of O_i operations in time to discover s. Simple SCAs (SSCAs) can be easily mounted in a single leakage trace (e.g., in RSA of ECC (Error Correction Codes) implementation) and are typically horizontal type of attacks meaning that they are mounted using a single leakage trace processed in time. When SSCAs are not possible, advanced SCAs (ASCAs) must be mounted to extract s [36].

Advanced SCAs do not focus only on the operations (e.g., O_i) but also on the Computation operands [36,37]. Advanced SCAs are focused on a subset of the calculation C (and/or O_i) and through collection of sufficiently large number N of leakage traces $L_i(t)$ for all t in {1, ..., N} using inputs $X_i(t)$ exploit the statistical dependency between the calculation on C for all X_i and the secret s. ASCAs follow the hypothesis test principle [36,37] where a series of hypothesis s' on s (usually on some byte or bit j of s i.e., $s_j{}' = 0$ or 1) is made and a series of leakage prediction values are found based on each of these hypothesis using an appropriate prediction model. The values of each hypothesis are evaluated against all actual leakage traces using an appropriate distinguisher Δ for all inputs X_i so as to decide which hypothesis is correct.

SSCAs and ASCAs can follow either the vertical or horizontal leakage collection and analysis strategies. In the vertical approach, the implementation is used N times employing either the same or different inputs each time t in order to collect traces-observations $L_i(t)$. Each observation is associated with t-th execution of the implementation. In the horizontal approach, leakage traces-observations are collected from a single execution of the implementation under attack and each trace corresponds to a different time period within the time frame of this execution. As expected, in Horizontal attacks the implementation input is always the same. ASCAs are also called and Differential SCAs.

The distinguisher used in order to evaluate the hypothesis against the actual leakage trace is usually a statistical function like Pearson correlation or collision correlation.

5. Countermeasures

5.1. Rowhammer Attacks Countermeasures

There exist various countermeasures for rowhammer attacks proposed in the literature. In [5] various system level mitigations such as enforcing Error Correction Codes (ECC) protection, increasing the refresh rate of DRAM cells, retiring DRAM cells that the DRAM manufacturer identifies as victim cells and refreshing vulnerable rows were proposed.

With the exposure-publication of the rowhammer vulnerability attack, specific software based solutions were introduced for mitigating the problem. The first such countermeasure was the disallowance or rewriting of the flush instructions, that have been deployed in Google NaCl. Also, on the new kernel Linux, the pagemap interface is prohibited from userland so as to block relevant rowhammer attacks [19]. However, new attacks have appeared that overpassed the above countermeasure, by providing rowhammer triggering using JavaScrip [17] or native code [11], without needing cache eviction. Similarly, the pagemap based countermeasure has been bypassed by the use of different methodologies of finding out the machine address map to a physical address or the map to the virtual userland [8,14,16].

Furthermore, some hardware manufacturers have implemented a series of measures to reduce the probability of a successful attack. However, the solution of providing memories with ECC does not offer strong security guarantees [6] because ECC cannot detect multiple bit flips on the same row in order to correct them. Most of the hardware based countermeasure just double the DRAM refresh rate however even then the probability of bit flips is still high enough to be exploited in some machines because the rate will need to be increased up to eight times to achieve a competent mitigation [5].

There also exist more generic countermeasures that try to detect rowhammer by monitoring the last-level cache misses on a refresh interval and row access with high temporal locality on Intel/AMD processors [12]. If the missing cache overpasses a threshold, a selective refresh is performed on the vulnerable rows.

Other countermeasures rely on detection of activation patterns. Following this direction, in [5], the PARA (Probabilistic Adjacency Row Activation) mechanism is implemented in the memory controller. This countermeasure's approach is to refresh the adjacent row of the accessed row with a probability p each time (acts not deterministically) thus avoiding the hardware costly complex data structure for counting the number of row activations to mitigate the rowhammer attack. Similarly, another memory controller implementation solution is ARMOR [38] that includes a cache buffer for frequently activated rows.

DDR4 memory modules are harder to rowhammer since they include Targeted Row Refresh (TTR) techniques in the form of a special module tracking the row activation frequency so as to select vulnerable rows for refreshing [39]. The LPDDR4 standard also incorporates TTR as well as the Maximum Active Count (MAC) technique [40]. Yet still, as noted in [8] the rowhammer vulnerability bug exists in DDR4.

When there is a need for a high-security level in embedded system devices (e.g., in critical infrastructures) the above-described countermeasures are not enough to guarantee this high level. More specialized solutions must be developing for each kind of architecture, OS and usage in order to shield the devices from the exploited vulnerabilities used in the existing rowhammer attacks. The first step in this process is to develop a mechanism to disable memory deduplication.

5.2. Side Channel Analysis Attack Countermeasures

As was described above, SCAs can be very potent if appropriate countermeasures are not introduced in the device under attack. All SCAs rely on some side channel information leakage, so SCA countermeasures aim at either hiding this leakage or at minimizing this leakage so that it contains very small or no useful information to mount an attack (making the leakage trace, data independent) [41]. The most widely used countermeasure, primarily applicable on microarchitecture attacks, is making the security operation time delay constant or random, regardless of the microarchitecture elements that are used or the implemented code it is executed [42]. However, developing constant-time execution code is not always easy since optimizations introduced by the compiler must be bypassed. For this reason, specialized constant-time libraries have been made available in the literature, to help security developers protect their applications against SCAs. Similar to constant time implementations, effort has also been invested to structure power and EM constant approaches. Such approaches follow algorithmic modifications of the cryptography functions realized either in software or in hardware [36]. In the latter case, circuit based modifications can also be made so as to thwart SCAs (e.g., Dual Rail Technology, special CMOS circuits) [43]. The goal of these countermeasures is making the leakage trace of O_1 indistinguishable from the leakage trace of O_0. Hardware support has also been introduced in certain processor technologies (e.g., Intel processors) to make constant-time cryptography operations (e.g., for AES).

For microarchitectural/Cache SCAs, cache specific countermeasures have been introduced in literature. A good overview of the countermeasure directives is made in [38] and [24]. Typically, countermeasures can be divided in isolation measures and randomization measures. By isolation, the attacker no longer shares cache with the victim so external interference related attacks become very difficult. Isolation can be achieved by splitting cache into zones related to individual processed for example using hardware virtualization. This partitioning can be done either statically or dynamically. Randomization is a more generic countermeasure that is applicable in most of the SCAs. In practice, it can be supported by random cache eviction or random cache permutation mechanisms [24].

In general, the provided solutions, try to nullify the cache characteristics that cache SCAs exploit (e.g., FLUSH + RELOAD, PRIME + PROBE and EVICT + TIME). Among the existing solutions (fitted in the above-mentioned categories) there exist techniques to eliminate timing side channels by introducing virtual time and black-box mitigation techniques [42]. Also, time partitioning can be used through regular Cache Flushing, Lattice Scheduling, Memory Controller Partitioning, structuring Execution Leases and performing Kernel address space isolation. Finally, hardware partitioning is a proposed countermeasure approach that is focused on disabling hardware threading, page sharing, introducing Hardware Cache Partitions, quasi-partitioning even migrating VMs of cloud services. There also exist approaches that by modeling a cache memory's side channel leakage, they can evaluate a system's vulnerability to specific cache side channel attack categories [24] thus constituting a proactive cache SCA countermeasure.

It must be mentioned here, that some of those countermeasures can be bypassed using the rowhammer attacks using the approaches described in previous subsections.

Apart from cache SCAs countermeasure, randomization is a favorable solution for countering all ASCAs (both horizontal and vertical). Using randomization, the sensitive information is disassociated from the leakage trace and is hidden by multiplicatively or additively blinding this information using a randomly generated number. A variant of this technique is information hiding where random noise is added to the leakage channel thus scrabbling making the associate to the leakage trace unusable [36].

6. Conclusions

In this paper, a survey was made on malicious attacks that can be mounted on real-time embedded system devices due to computer architecture or hardware vulnerabilities and side channel leakage exploitation. Such attacks use software code that does not directly interfere with other software tools (infect programs, directly alter their functionality) but rather exploit intrinsic vulnerabilities on the embedded system device hardware to give leverage to an attacker. In this paper, we focus on the recently exploited rowhammer vulnerability and on SCAs attacks (microarchitectural/cache attacks and power/EM analysis attacks) and provide an overview of the attack research directions highlighting their applicability in the various computer architectures. The attacks' analysis performed in this paper indicate that the mentioned attacks can reduce the QoS level of an embedded system as well as defeat many security and trust enhancing technologies including software cyberattack tools (antimalware, anomaly detection, firewalls) but also hardware assisted technologies like memory isolation through VMs, Trusted Execution Environments, ARM TrustZone or dedicated security elements. The real-time embedded system security architect must be informed of this attack taxonomy and must put pressure on embedded system device manufacturers to provide security patches to their products, capable of thwarting the attacks.

Acknowledgments: The work in this paper is supported by CIPSEC EU Horizon 2020 project under grant agreement No. 700378.

Author Contributions: Apostolos P. Fournaris conceived and structured the paper's concept, did the overview on cache35 attacks, side channel attacks and countermeasures and contributed to the survey research of the rowhammer microarchitectural attacks, overviewed the whole text and wrote Sections 4 and 5.2. Lidia Pocero Fraile did the overview research for the Rowhammer attack and was responsible for Sections 2 and 3. Odysseas Koufopavlou overviewed the research process, contributed to the paper's concept and overall text formation.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Enhancing Critical Infrastructure Protection with Innovative SECurity Framework (CIPSEC). H2020 European Project. Available online: www.cipsec.eu (accessed on 28 March 2017).
2. Challener, D.; Yoder, K.; Catherman, R.; Safford, D.; Van Doorn, L. *A Practical Guide to Trusted Computing*; IBM Press: Indianapolis, IN, USA, 2007.
3. Trusted Computing Group. *TCG TPM Specification Version 2.0*; Trusted Computing Group: Beaverton, OR, USA, 2014.
4. ARM. ARMTrustZone. Available online: https://www.arm.com/products/security-on-arm/trustzone (accessed on 1 April 2017).
5. Kim, Y.R.; Daly, J.; Kim, C.; Fallin, J.; Lee, H.; Lee, D.; Wilkerson, C.; Lai, K.; Mutlu, O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), Minneapolis, MN, USA, 14–18 June 2014.
6. Lanteigne, M. How Rowhammer Could Be Used to Exploit Weakness in Computer Hardware. 2016. Available online: https://www.thirdio.com/rowhammer.pdf (accessed on 1 April 2017).
7. Van der Veen, V.; Fratantonio, Y.; Lindorfer, M.; Gruss, D.; Maurice, C.; Vigna, G.; Bos, H.; Razavi, K.; Giuffrida, C. Drammer: Deterministic rowhammer attacks on mobile platforms. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria, 24–28 October 2016.
8. Pessl, P.; Gruss, D.; Maurice, C.; Schwarz, M.; Mangard, S. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In Proceedings of the USENIX Security Symposium, Austin, TX, USA, 10–12 August 2016.
9. Bosman, E.; Razavi, K.; Bos, H.; Giuffrida, C. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In Proceedings of the 2016 IEEE Symposium on Security Privacy, SP 2016, San Jose, MA, USA, 23–25 May 2016; pp. 987–1004.

10. Seaborn, M.; Dullien, T. Exploting the DRAM rowhammer bug to gain kernel privileges. In Proceedings of the 2016 ACM SIGSAC Conference, Vienna, Austria, 24–28 October 2016.
11. Qiao, R.; Seaborn, M. A new approach for rowhammer attacks. In Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 3–5 May 2016.
12. Aweke, Z.B.; Yitbarek, S.F.; Qiao, R.; Das, R.; Hicks, M.; Oren, Y.; Austin, T. ANVIL: Software-based protection against next-generation rowhammer attacks. In Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Atlanta, GA, USA, 2–6 April 2016.
13. Moinuddin, K.Q.; Dae-Hyun, K.; Samira, K.; Prashant, J.N.; Onur, M. AVATAR: A variable-retention-time (vrt) aware refresh for dram systems. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, 22–25 June 2015.
14. Program for Testing for the DRAM Rowhammer Problem. 2015. Available online: https://github.com/google/rowhammer-test (accessed on 15 March 2017).
15. Seaborns, M. How Physical Adressesses Map to Rows and Banks in DRAM. 2015. Available online: http://lackingrhoticity.blogspot.com/2015/05/how-physical-adresses-map-to-rows-and-banks.html (accessed on 5 April 2017).
16. Xiao, Y.; Zhang, X.; Teodorescu, R. One bit flips, one cloud flops: Cross-VM row hammer attacks and privilage escalation. In Proceedings of the 25th USENIX Security Symposium, Austin, TX, USA, 10–12 August 2016.
17. Gruss, D.; Maurice, C.; Mangard, S. Rowhammer.js: A remote software-induced fault attack in javascript. In Proceedings of the 13th Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA), Donostia-San Sebastián, Spain, 7–8 July 2016.
18. Bhattacharya, S.; Mukhopadhyay, D. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. *Lect. Notes Comput. Sci.* **2016**, *9813*, 602–624.
19. Salyzyn, M. AOSP Commit 0549ddb9: UPSTREAM: Pagemap: Do Not Leak Physiscal Addresses to Non-Privilahe Userspace. 2015. Available online: http://goo.gl/Qye2MN (accessed on 1 May 2017).
20. Razai, K.; Gras, B.; Bosman, E.; Preneel, B.; Giuffrida, C.; Bos, H. Flip feng shui: Hammering a needle in the software stack. In Proceedings of the 25th USENIX Security Sympoisium, Austin, TX, USA, 10–12 August 2016.
21. Joye, M.; Tunstall, M. *Fault Analysis in Cryptography*; Springer: New York, NY, USA, 2012.
22. Irazoqui, G.; Eisenbarth, T.; Sunar, B. Cross processor cache attacks. In Proceedings of the 2016 ACM Asia Conference Computer Communications Security, Xi'an, China, 30 May–3 June 2016; pp. 353–364.
23. Zhang, N.; Sun, K.; Shands, D.; Lou, W.; Hou, Y.T. *TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices*; Cryptology ePrint Archive Report 2016/980; The International Association for Cryptologic Research (IACR): Cambridge, UK, 2016.
24. Zhang, T.; Lee, R.B. *Secure Cache Modeling for Measuring Side-Channel Leakage*; Technical Report; Princeton University: Princeton, NJ, USA, 2014; Available online: http://palms.ee.princeton.edu/node/428 (accessed on 3 March 2017).
25. Wang, Z. Information Leakage Due to Cache and Processor Architectures. Ph.D. Thesis, Princeton University, Princeton, NJ, USA, 2012.
26. Bernstein, D.J. Cache-Timing Attacks on AES. 2005. Available online: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (accessed on 12 April 2017).
27. Percival, C. Cache missing for fun and profit. In Proceedings of the BSDCan 2005, Ottawa, ON, Canada, 13–14 May 2005.
28. Osvik, D.A.; Shamir, A.; Tromer, E. Cache attacks and countermeasures: the case of AES. In *Topics in Cryptology–CT-RSA 2006*; Springer: New York, NY, USA, 2006; pp. 1–20.
29. Aciicmez, O.; Brumley, B.B.; Grabher, P. New results on instruction cache attacks. In Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems, Santa Barbara, CA, USA, 17–20 August 2010; pp. 110–124.
30. Yarom, Y.; Falkner, K. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), Santa Barbara, CA, USA, 20–22 August 2014; pp. 719–732.

31. Gullasch, D.; Bangerter, E.; Krenn, S. Cache games–bringing access-based cache attacks on AES to practice. In Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 22–25 May 2011; pp. 490–505.
32. Liu, F.; Yarom, Y.; Ge, Q.; Heiser, G.; Lee, R.B. Last-level cache side-channel attacks are practical. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 605–622.
33. Irazoqui, G.; Inci, M.S.; Eisenbarth, T.; Sunar, B. *Wait a Minute! A Fast, Cross-VM Attack on AES*; Lecture Notes Computer Science; Springer: New York, NY, USA, 2014; pp. 299–319.
34. Zhang, X. Return-oriented flush-reload side channels on arm and their implications for android security. In Proceedings of the 2016 ACM SIGSAC Conference on Computer Communications Security (CCS'16), Vienna, Austria, 24–28 October 2016; pp. 858–870.
35. Lipp, M.; Gruss, D.; Spreitzer, R.; Maurice, C.; Mangard, S. ARMageddon: Cache attacks on mobile devices. In Proceedings of the 25th USENIX Security Symposium, Austin, TX, USA, 10–12 August 2016; pp. 549–564.
36. Fournaris, A.P. Fault and power analysis attack protection techniques for standardized public key cryptosystems. In *Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment*; Sklavos, N., Chaves, R., di Natale, G., Regazzoni, F., Eds.; Springer: Cham, Switzerland, 2017; pp. 93–105.
37. Bauer, A.; Jaulmes, E.; Prouff, E.; Wild, J. Horizontal and vertical side-channel attacks against secure rsa implementations. In *Topics in Cryptology—CT-RSA 2013: The Cryptographers' Track at the RSA Conference 2013*; Dawson, E., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 1–17.
38. Ghasempour, M.; Lujan, M.; Garside, J. ARMOR: A Run-Time Memory Hot-Row Detector. 2015. Available online: http://apt.cs.manchester.ac.uk/projects/ARMOR/Rowhammer (accessed on 6 March 2017).
39. *DDR4 SDRAM MT40A2G4, MT401G8, MT40A512M16 Datasheet, 2015*; Micro Inc.: Irvine, CA, USA, 2015.
40. JEDEC Solid State Techonlogy Association. *Low Power Double Data Rate 4 (LPDDR4)*; JEDEC Solid State Techonlogy Association: Arlington, VA, USA, 2015.
41. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*; Springer: New York, NY, USA, 2007.
42. Ge, Q.; Yarom, Y.; Cock, D.; Heiser, G. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptogr. Eng.* **2016**, 1–27. [CrossRef]
43. Tiri, K.; Verbauwhede, I. A digital design flow for secure integrated circuits. In Proceedings of the IEEE Transactions CAD Integrated Circuits System, Pennsylvania, PA, USA, 12–15 August 2006; pp. 1197–1208.