POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN COMPUTER SCIENCE AND ENGINEERING

GLOBAL PROTECTION FOR TRANSIENT ATTACKS

Doctoral Dissertation of:
Niccolò Izzo

Supervisor:
Prof. Luca Oddone Breveglieri

Tutor:
Prof. Luciano Baresi

The Chair of the Doctoral Program:
Prof. Luigi Piroddi

2022 – Cycle XXXIV

Time moves in one direction, memory another.
We are that strange species that constructs artifacts
intended to counter the natural flow of forgetting.

William Gibson - Distrust That Particular Flavor

I would like to thank my advisor Luca, you have always supported my will and helped overcoming all the obstacles along the way.

Alessandro and Gerardo have all my gratitude, for showing me how to do sound scientific research, without ever accepting easy way outs.

Paolo, Marco, Danilo, Tamara and all the colleagues from Micron, you inspired me with the wholehearted passion you devote to technology.

Alessandro, Pietro, and all the people from Rev.ng, you taught me how to strive for technical excellence, and it was fun all along.

Thank you Federico for sharing your endless drive for self-improvement.

Marzia, I owe it to you if I'm still human, after all.

Finally thanks to all my bonds, my family, my friends, although we might be distant, we still shape our lives together.

All that I've learned through this PhD experience, I owe it to you.

Abstract

COMPUTER security threats have a strong bond with information permanence, which is encoded in the physical state of a device. Attacks based on the trace left by the flow of information into a system in the form of its physical state are called transient. A secure device must store its state in a multitude of functionalities that have to be resilient to known and future attacks. In a mobile system, the security state of the device could switch between locked and unlocked, and the secure erasure of user data must be guaranteed during said transitions. Current DRAM-based main memories will be gradually replaced by Emerging Memories such as 3D XPoint, ReRAM, STT-RAM, Memristor or ULTRARAM, which are faster, more scalable and efficient than traditional NAND flash, even though their non-volatility is yet another potentially vulnerable state. Thus, a secure non-volatile storage architecture will have to employ well-known cryptographic building blocks to guarantee strong security properties on the stored data, such as confidentiality, integrity and replay protection, even when the device is turned off. Such properties must be guaranteed despite external physical threats, tampering with the bus signals, as well as internal threats, executing malicious code in a Virtual Machine on the same virtualized environment, or on the hypervisor itself. Another threat that originates from the variation of physical states are side-channel attacks. In fact, even the most efficient encryption architecture is rendered useless if a secret, e.g., a cryptographic key, is exposed through side-channel leakage, like power consumption, EM emission, or others. Masking techniques allow to implement effective software countermeasures, however their security proofs can be invalidated by hidden micro-architectural features. To restore the effectiveness of these countermeasures, a detailed model of the stateful elements of the data path has to be derived. Such model will allow the modification of the instruction scheduling of the sensitive code to implement side-channel countermeasures, e.g., masking, in a secure way.

Contents

CHAPTER 1

Introduction

Computer systems encode and process information through the variation of physical states of a multitude of components. These physical states are propagated to other components, potentially of different nature by using transducers and interconnections. As an example, the charge of a Dynamic Random Access Memory (DRAM) capacitor (cell) stores a single logic bit. Upon a read operation, such charge level is sensed by an analog amplifier and is transported onto the voltage (and thus the corresponding logical level) of a single DQ wire of a DDR bus [79]. The value of such wire is sensed by the PHYsical (PHY) layer transducer of the host memory controller and is forced into the physical state of the flip-flop that represents a particular bit of a CPU register.

One way of interpreting the main purpose of Computer Security is the control of access, in terms of reading and writing capabilities, to the information encoded in such physical states. According to the specific nature of a computing system, such purpose translates into the design and validation of architectures, that ensure the impossibility of diverting such information flow into a rogue system, e.g., through the implementation of encryption strategies, or through explicit filtering hardware, able to allow or deny access to a resource according to the privilege level, or lastly if the attack model is the side channel inference, by employing mitigations to the side channel-based information extraction. In the case of write attempts, the impossibility of manipulation of the physical state of the system must be guaranteed, and is typically performed through the implementation of cryptographic integrity algorithms.

In the following sections and in the embedded works, we will present four different instances of the same problem of controlling access to information. The first instance [77] will be the protection of the information being transferred on a physical memory bus against protocol sniffing and tampering attempts. The second instance [23] will cover the protection of a cryptographic secret from an Electro-Magnetic (EM) [61] Correlation Power Analysis (CPA) [31] attack, the third work will cover the information access control among virtual software entities, co-existing on the same hardware and sharing one or more physical links, finally, the fourth work will discuss a strategy for securely managing state transitions in mobile security architectures.

Each effort to control information access, always bears a cost. A complementary goal of computer security is to lower that cost such that the performance of the computing system to be protected is preserved. Research efforts in this field explore the trade-off between security and costs to achieve practical security with bearable performance drawbacks.

1.1 Attacks on the Main Memory

Data contained in the main memory of a system on a given instant can be defined as *data in use*, and represents a snapshot of the computation being performed in that system. Data in use can contain valuable information such as cryptographic keys, certificates, personal data and intellectual property. Confidentiality and integrity of the main memory are necessary to the trustworthiness of such a system. In fact, if the confidentiality of the main memory is broken, not only the confidentiality of the current working data might be immediately violated, but privilege escalation attacks might happen and threaten the security of all the data that will transit through the system in the future. One example of such a case is the Heartbleed [143] vulnerability, in which the OpenSSL [147] library leaked portions of the main memory to an attacker, and this led to a complete system exploitation.

When the main memory is no longer confidential, for example as a consequence of a cold boot attack [68], also the confidentiality of cold data, even if stored in encrypted mass storage devices can be compromised. In fact, during the normal usage of a system, disk encryption keys reside in cleartext form in main memory and can be easily fingerprinted, extracted, and exploited to access all the data stored in the computing system [68]. Mitigations to cold boot attacks like TRESOR [107] and Loop-Amnesia [138] try to hide the encryption key outside of the system memory, respectively in x86 CPU debug registers [107] or in AMD64 or EMT64 profiling registers [138]. But if memory integrity is also violated, these coutermeasures become ineffective, e.g., against an attack that is performed interactively, through the use of an interposer and a logic analyser. In fact both mitigations leverage CPU registers to store the disk encryption keys, and in both cases these might be leaked if the attacker induces the system to execute a binary that the attacker copies in the Random Access Memory (RAM) itself, e.g., overwriting a component of the host kernel. Even a violation of memory integrity alone, can lead to privilege escalation and data exfiltration, because all the software that we execute, including software-based security implementations, assumes that the main memory does not lie. For this reason, main memory tampering is a desirable target from an attacker standpoint; and for the same reason the Rowhammer [85] class of vulnerabilities is so dangerous, as it is able to corrupt (write) memory by only issuing memory read commands. Even though the attacker might not be able to see a cleartext copy of the data, he can still perform blind attacks to achieve privilege escalation. As an example, he might generate a large number of Page Table Entries (PTE) in the system memory and try to corrupt the pointer of one of these PTEs to gain write access to an attacker controlled PTE, and then pointing that PTE to the kernel memory, and gaining write access to kernel memory [123]. Another integrity-only attack could aim at corrupting an RSA key to weaken its cryptographic properties and make it more easy to factorize as demonstrated by Razavi et. al. [126].

A primary road to walk while attempting main memory readout and corruption is the one of physical attacks. Such attacks might for example try to sniff data that travels on the main memory bus, or to issue rogue transactions to the memory, to perform a complete readout of the memory, or even to tamper with the memory protocol employed by the main memory modules, for example trying to roll back transactions, restoring an old state of the memory, e.g., to roll back a critical software update [39]. As formalized by the Common Criteria [42] analysis framework, the danger of an attack is directly proportional to the cost of the equipment involved. Main memory busses typically operate with frequencies on the order of several gigahertz, as an example, the LPDDR4 bus is able to reach 3.2GT/s [80]. Logic analyzers able to sample such frequencies are bulky and costly, therefore physical attacks on the main memory bus have for a long time been considered unfeasible. Recently however, FPGA-based memory-specific analyzers like the Antmicro LPDDR4 Test Platform [15] were produced. Hardware such as the Antmicro significantly lowers the cost barrier to perform sniffing and tampering attacks on the memory bus. Even though this problem can be generally solved with the use of End-to-End (E2E) encryption, there are cases that require the peripheral to have access to plaintext data. Among such cases are Processing In Memory (PIM) [65] and Near Memory Processing [108] applications, where the accelerator needs to perform computation on the data to achieve higher efficiency. Other cases are represented by memory devices which are based on emerging memories like Memristors [124], Spin-Transfer Torque (STT-RAM) [84], ULTRARAM [91], Phase Change Memory (PCM) [128] and 3D XPoint (3DXP) [106]. Such emerging memories, when fed with cleartext data can leverage optimized write strategies to reduce the medium wear-out. These premises combine into the need for an effective link-encryption solution for memory protocols.

1.2 From Physical to Virtual Security

The latest datacenter and hyperscaler platforms make a heavy use of virtualization technology. This technology leverages a HW/SW component called Virtual Machine Manager (VMM) or Hypervisor to abstract the available resources and execute one or more Virtual Machines. The advantage of this technique is manifold, the hardware resources are used more efficiently: since a customer Virtual Machine might not use all the available CPU time, several Virtual Machines can use a larger portion of the available computational power. Other advantages of virtualization include the possibility to perform a snapshot of a Virtual Machine for backup purposes, and to migrate or clone the VM, increasing the deployment flexibility. This great scalability is not shared by traditional memory channels like DDR5 [79], which are not hot-pluggable and are not suitable for emerging memory due to their lack of support to persistent memory semantics. However, a new protocol that is gaining popularity aims at solving these issues; called Compute eXpress Link (CXL) [137], is a CPU interconnect protocol, based on the PCI express 5.0 PHY [116]. CXL can be used to interconnect external main memory modules, or memory-equipped accelerators, in a cache-coherent way, is hot-pluggable, and leverages all the flexibility of its PCIe PHY, e.g., like the possibility to set up a CXL switching fabric, to accompany the scalability features of emerging technologies.

Since the publication of our article on LPDDR4 link level security [77], several steps ahead were made in standard implementations for memory link protection. The CXL Integrity and Data Encryption (IDE) [137], represents one notable example of link-level security which is integral part of the standard. CXL IDE inherits largely from PCIe IDE [117], providing confidentiality, integrity and replay protection to all the protocol flits (PHY packets), through the usage of parallel AES-GCM encryption/decryption sessions. CXL IDE uses ephemeral keys, which are programmed through an encrypted protocol called Security Protocol and Data Model (SPDM) [54], which adopts a certificate-based attestation scheme. In current CXL standard, however, attestation can only be provided at physical port granularity, which means that the protocol itself is not able to discriminate between the single virtual entities in execution on a host platform; either the whole host and all of its Virtual Machine are trusted, or none of them are. Thus, a separation between VMs or between Virtual Machine Manager and VMs is completely absent. These reasons motivate the need for a solution to enforce access control beyond the physical port granularity. Such a solution should allow to preserve the confidentiality, integrity and replay protection properties with the granularity of a single virtualized entity.

1.3 Mobile Devices Security

Mobile devices are equipped with sophisticated security architectures, based on a Chain of Trust (CoT) mechanism, able to enforce a cryptographic validation chain on all the executed software. The Chain of Trust is anchored in the device immutable Read Only Memory (ROM), which is part of the System on Chip (SoC) silicon mask, and contains the first code that is executed on the device, which is de-facto immutable. The ROM code measures and executes the following software layer, that is in charge of measuring and executing the next and so forth. This validation chain is composed by primary and secondary bootloaders, which execute the operating system kernel. The kernel itself, maintains a Merkle tree over the system partition, that contains the executed software. This mechanism achieves security at the expense of the user freedom, in fact, as this validation mechanism is signed by the Original Equipment Manufacturer (OEM), the final users of the device do not possess the keys to run arbitrary software on devices they own. To remediate this situation, most of the mobile Original Equipment Manufacturers (OEM), provide a mechanism to switch the device into an *unlocked* state, in which the cryptographic signature of the kernel is disabled to allow the user to run arbitrary third-party software, including alternative operating systems like LineageOS [146] and PostmarketOS [121]. However the same device unlock, if performed by an evildoer, can lead to a complete compromise of the device security and exfiltration of all the user data contained therein, or worse, to the implantation of a covert, permanent rootkit, which can remain undetected for a long time, allowing the continuous collection of user data, in a scenario which is commonly defined as Advanced Persistent Threat (APT). The Android ecosystem, is characterized by a wild variety of System on Chips (SoC) and device manufacturers, with each combination implying a different design of the unlock mechanism. Unfortunately, the lack of a unified, well scrutinized unlock mechanism, instead leaves room to many vendor-specific implementations which are often vulnerable. Therefore there is the need for a publicly documented, cryptographically provable solution, to allow the final user, with the OEM consent, to be able to unlock its own device, with the guarantee of the erasure of the user's own sensitive data. Finally, the unlock procedure should be tamper evident, such that the final user can notice if the procedure was applied to his device without his consent. The only hardware prerequisite for setting up such a mechanism is the availability of an integrity protected state storage mechanism, which can maintain knowledge about the security state of a device.

1.4 The Devil is in the Microarchitectural Details

Logical lines commute from one binary logical value to the opposite one through a variation of electrical field, which in turn generates an Electro-Magnetic (EM) radiation that diffuses from the activated components, to the outside of the physical boundary of the device, and can be sensed by external equipment. As a consequence, all the electronic circuits emit an Electro-Magnetic radiation which is correlated with the Hamming Distance (HD) of the logical values which evolve on all the latches and data lines of the circuit itself [50]. Such emissions typically have a small magnitude, and can be subdued by stronger signals emitted by other portions of the circuit, like switching voltage regulators, NOR/NAND flash erase-write cycles and background Electro-Magnetic noise. However, if a portion of the processed data is kept constant for a large enough number of iterations, which is typically the case for cryptographic keys, the leakage of constant data can be considered as a signal, and the rest of the emissions can be considered as noise [70]. If an attacker is able to collect a large number of measurements, statistical methods can be applied to increment the Signal to noise Ratio (SnR), and make cryptographic key material emerge from the noise floor generated by the rest of the circuit.

Throughout the years, several side channel attack countermeasures have been proposed. Based on their working principle, they can be divided into three categories: hiding [11, 24, 103, 135], which generates additional noise to further reduce the SnR, masking [37, 130] which mixes the secret with random values re-generated at each cipher iteration to hamper the statistical data processing, and morphing [8–10, 16] which reshapes the encryption or decryption algorithm to invalidate the attacker's assumptions on the cipher structure and Hamming distance models, making the cipher a moving target. If we consider a masking implementation of order d, for each iteration of a cipher, a set of $d >= 1$ random values called *masks*, is used to break each sensitive value into $d + 1$ *shares*. Each masking scheme provides both a method to split a value into *shares* and a set of methods to compute arbitrary Boolean functions on share-split values [129]. If the correlation of the *shares* is never processed (and thus never leaked), the masking scheme can be proven to be secure against statistical methods of order d [130], given a particular measurement model called *probing model* [129]. However guaranteeing that the *shares* are never mixed in software implementations can be harder than it seems, since the programmer only has visibility at Instruction Set Architecture (ISA) level.

On all the commercial architectures, a detailed hardware description of the processor is not available, and the programmer is only given a set of functional guarantees about the execution of assembly instructions. In fact, the underlying hardware can silently re-combine values which are independent from the ISA perspective [25]. A detailed hardware description of the processor thus becomes a critical need, which can only be retrieved through reverse engineering of the architectural details of the CPU microarchitecture. In fact the knowledge of the micro-architectural features of a CPU can be either exploited to break software masking schemes which are unaware of such features, or be used to build software masking schemes which are aware of such features and thus secure against the aforementioned newly designed exploits [62, 155]. The micro-architectural features that we have interest in characterizing from this perspective are many: from the issue-width, to the number of pipeline stages, to the type and number of functional units and the presence of synchronization buffers therein. Each of these features has an impact on the system performance and on the device EM leakage. In the second work embedded in this thesis [23], we designed a framework to exploit such variations for characterizing hardware, enabling the setup of more sophisticated attacks, as well as the design of better software masking countermeasures.

CHAPTER 2

State of the Art

In this section, for each of the three introduced research areas: physical attack on main memory buses, attacks between virtualized entities coexisting on the same Virtual Machine Manager, and microarchitectural side channel methods to characterize and exfiltrate the content of CPU datapath registers, we will provide a comparison with the state of the art in both academic and industrial works. This section has the double purpose of offering the information on the setting of the problem that we are presenting, as well as presenting commercial and academic solution of the problem, with the eventual associated shortcomings and limitations.

2.1 A Secure and Authenticated Host-to-Memory Communication Interface

To better understand the problem statement that led to the development of our work, here we propose a brief history of the evolution of memory encryption techniques, from their early commercial implementations, to the most sophisticated extensions that aim for a secure operation of Virtual Machines.

2.1.1 The Game of Hacking the Xbox

The history of memory encryption is strongly intertwined with the one of Digital Rights Management (DRM) and anti-piracy. One of the first academic examples of such a system was exposed by Lie et. al. [92] and has the explicit purpose of preventing the unauthorized copy of software. One of the application fields in which anti-piracy is particular relevant is the videogame industry. Videogame consoles are often sold at loss, a debt which is paid back with the margins on the title sales. Game piracy can significantly reduce such margins, and therefore it is fought at all costs by videogame console makers [72]. It does not come as a surprise that the first widespread examples of memory encryption can be found in videogame consoles.

The Xbox gaming console by Microsoft, used a symmetric-key encrypted ROM, whose key was read from the southbridge bus. Eventually the key was recovered by sniffing it while it was transmitted on the bus during system boot [72, 141]. The compromission of the Xbox security led to the growth of a mod chip industry for that console and to loss of revenue from Microsoft. Microsoft reacted in the following generation of their console by encrypting the whole system memory [47]. In the case of the Xbox 360, Microsoft was likely willing to pay an extra performance and complexity cost for encrypting memory, since the piracy prevention was at stake, as well as protection against cheaters, that could have undermined the experience of Microsoft's own online gaming platform *Xbox Live*. However in the security architecture of the Xbox 360, the encryption and hashing of main memory were selected by the upper 32 bits of the 64-bit addresses. An attacker could disable the encryption feature by jumping to an address which aliases with the kernel memory and has the encryption and hashing bits cleared, which led to the complete exploitation of the system [105]. The lessons learned from the Xbox cases provided the drive to research new memory encryption mechanisms, controllable by the operating system [69], and operating directly on the main memory bus [142].

2.1.2 Performance Considerations

Even though memory encryption is an effective solution to protect information confidentiality, effectively slowing down the reverse engineering of a system and the search for its vulnerabilities, as well as protecting memory integrity, to prevent the system corruption and privilege escalation, however it comes at a cost. As an example, the performance impact of the XOM security architecture proposed by Lie et. al. [92], which takes 48 cycles to encrypt a cache line, was estimated by Yang et. al. to increase the execution time of software by up to 35% [154]. Since the performance penalty arises from an increase of latency during the cache line read, the penalty can be reduced if the cache line is prefetched and becomes available in the cache at execution time. Such a solution was proposed by Rogers et. al. [132] and led to a mitigation of the decryption latency, at the cost of causing unnecessary decryptions due to the erroneous prefetcher predictions.

2.1.3 Encryption in Contemporary CPUs

In the last decade however, memory encryption techniques are being introduced also in commodity computing system, often starting from server-grade CPUs such as the EPYC series from AMD [6], where the occupied silicon area is already large and the added area cost of implementing the new technology is sustainable. Furthermore the datacenter environment, especially in the case of hyperscaler companies is less chaotic with respect to the mass market, and usually totally or partially under the control of the datacenter company, which makes the introduction of a new feature easier from an ecosystem standpoint. As an example, if a feature requires a certificate infrastructure for attestation purposes, it might be easier by the datacenter owner to issue a certificate for each of the entity of the datacenter itself. Conversely, when a technology is publically deployed, either the certificates are emitted by a centralized Certification Authority, as it happens with the World Wide Web TLS certificates [149], or a de-centralized approach is needed and manual trusting or Trust On First Use (TOFU) policies [151] must be put in place, as it happens with the GNU Privacy Guard (GPG) [144] email encryption system. While a decentralized approach is in general more privacy friendly, it is definitely harder to embrace given the diversity of the platforms which the system must support.

2.1.4 Application-Level Memory Encryption

One of the the first commodity encryption systems was Intel Software Guard eXtensions (SGX) [46] introduced in 2015 in all the 5th generation (codenamed Skylake) Intel Core CPUs. Intel SGX enables the creation of one or more *software secure enclaves*, these entities allow to perform arbitrary computations with integrity and confidentiality properties guaranteed over the processed data, in a system where the rest of the software, including Operating System (OS) and the Hypervisor are considered untrusted and potentially malicious. Enclaves are programs which have to be linked against Intel's own SGX Software Development Kit (SDK). Intel SGX relies on a trusted component on the server which executes the *secure enclaves*, this component is the Intel Management Engine: a Minix OS instance running in an ARC core inside the main CPU silicon [73]. Integrity and confidentiality of data in the DRAM are guaranteed through the encryption of the memory pages belonging to the *enclave*. Intel SGX leverages a Memory Encryption Engine (MEE) to encrypt the DRAM content. The MEE, as described by Gueron [66] uses a Merkle tree construction for guaranteeing integrity and a modified AES operating mode with a Carter-Wegman [35, 151] MAC for confidentiality. Intel SGX can encrypt the memory content of one or more *software enclaves*, but such an approach does not protect the whole system against hacking or tampering attempts. However the application itself is not only protected against external eavesdroppers sniffing data-in-use from main memory or its bus, but also from untrusted privileged software such as the kernel or an Hypervisor. The resulting trust boundary is very small and consequently has small attack surface. However this advantage has a significant complexity cost: SGX requires the software to be specially crafted to be compatible with its Application Programming Interface (API). For real world applications, the adaptation of existing software, often proves to be an infeasible option, either due to the high development cost, or the unavailability of the software source code, therefore mass-adoption of this technology is yet to be seen. SGX was deprecated in 2021 on consumer processors.

2.1.5 Host-level Memory Encryption

Just one year later, in 2016, AMD introduced a technology called Secure Memory Encryption (SME) [2] in their Zen CPU architecture, which includes: Ryzen desktop and mobile CPUs, Epyc server processors and Accelerated Processing Units (APU) for embedded applications. SME, conversely to its competitor technology SGX, encrypts the whole system memory, putting in the same security domain all the software that is running on a host, including the Hypervisor and Virtual Machines. The same technology is presently advertised under the name Memory Guard [3]. The encryption is performed by dedicated hardware AES engines, one for each main memory controller. The data is encrypted with a 128-bit key which is tweaked with the physical address of the accessed page to protect against ciphertext relocation attacks. The employed mode of operation however, is not publicly known. The 128-bit encryption key is re-generated randomly at each system boot and is not visible to any software running in the main CPU cores. Just like SGX, also SME requires a trusted portion of the system for managing the encryption keys, this role is held by the AMD Secure Processor (AMD-SP), formerly known as the Platform Security Processor (PSP), a 32-bit microcontroller (ARM® Cortex®-A5) embedded in the processor die [2] and running an undocumented, proprietary operating system [41].

Intel in 2017 released a white paper describing two technologies, respectively called Total Memory Encryption (TME) and Multi-Key Total Memory Encryption (MKTME) [75]. While TME is functionally equivalent to SME, save for the algorithmic choice of NIST standard AES-XTS algorithm with 128-bit or 256-bit keys derived from a hardware Random Number Generator (RNG). MKTME adds to that technology the possibility to manage up to 32'768 keys which can be configured by software entities. In the use case envisioned by Intel, an Hypervisor, upon creation of a new Virtual Machine, generates a new memory encryption key and configures that encryption key in the MKTME system such that the new Virtual Machine can have its virtual pages encrypted using that key. The advantage of this approach is that Virtual Machines do not need to be aware of the memory encryption technology and still can leverage its security benefits; conversely, the Hypervisor is a key trusted component in this architecture, thus it must be included in the trust boundary. Just as it happens for SGX, also TME and MKTME have their key creation and key storage and security logics implemented in the Intel Management Engine [73]. TME and MKTME will be available in 3rd generation Intel® Xeon® Processor Scalable Family processors, codenamed *Cooper Lake*, which will be likely released in 2022, targeting workstation and server systems.

2.1.6 VM-level Memory Encryption

Virtual Machine Managers are inevitably complex software stacks, mainly because they aim to support the creation of virtual machines using nearly or totally unmodified system images. Therefore they need to emulate a large portion of the hardware that can be found in todays' commodity machines. For example, the Windows 10 Operating system has no built-in support for virtual peripheral protocols like: SPICE [127] for video emulation or VirtIO for network, storage, serial and PCIe communication protocols [133]. Therefore, to be able to support and boot a vanilla Windows 10 image, an Hypervisor must be able to emulate a VGA graphics card, a SATA/SCSI disk, an e1000e Ethernet device or to provide direct access to real peripherals. The implementation of such a large amount of interfaces creates an equally large attack surface. One of the most used Hypervisors, likely due to its nature of Free and Open Source Software (FOSS) is QEMU-KVM. QEMU-KVM is the cooperation of two software components: QEMU [28] which is a user-mode and system-mode emulator, and KVM which is a kernel module that wraps hardware-assisted virtualization technologies like VT-x [76] and AMD-V [1] into a QEMU virtual CPU. Many attempts like rust-vmm [14], Enarx [57], gVisor [145] and Firecracker [7] can be used in place of QEMU to reduce the Virtual Machine Manager (VMM) attack surface at cost of losing legacy peripherals support and compatibility with existing systems. To be able to use existing feature-rich emulators, but keeping them outside of the trust boundary, we need specialized hardware technologies like AMD Secure Encrypted Virtualization (SEV) and AMD SEV-Secure Nested Paging (SEV-SNP).

AMD SEV [5,33] extends the SME security architecture by generating and employing a new AES symmetric key each time a Virtual Machine is booted. Conversely to MKTME, in SEV the keys are generated and managed entirely by the AMD Secure Processor. The Hypervisor is entirely excluded from the trust boundary, in fact the system is designed to be secure even with a semi-trusted Hypervisor. The AMD threat model [4, 5] defines as semi-trusted, an entity which is only relied upon for the availability of resources and not for their confidentiality. This means that if an Hypervisor is willing to terminate a Virtual Machine it will always have the power to do so, however if the Hypervisor wants to eavesdrop the data being computed inside a Virtual Machine, or stored in its main memory, it will not be able do to so if AMD SEV is employed. Buhren et. al. found a flaw in SEV's management of security keys which led to a complete circumvention of the security properties by an Hypervisor [33].

AMD SNP [4] is an extension to AMD SEV which provides integrity protection over the Virtual Machine memory pages, as well as protection against page aliasing, furthermore it patches the vulnerabilities found by Buhren et. al. [33]. SNP maintains a so called Reverse Map Table (RMT), which is a data structure under the control of the AMD SP, that keeps track, for each physical 4K memory block, of all the virtual pages which have been mapped to that block, furthermore it keeps an hash of the aforementioned block, keyed with the identifier of the entity that performed the last write. This architecture effectively protects against integrity violations attack, in which a victim Virtual Machine V protected by SEV-SNP stores its data in a main memory page, and an attacker VM A or the Hypervisor itself tries to corrupt the data therein contained, even without violating the confidentiality of the page itself. When V initially allocates the page, its identifier is written in the reverse mapping table entry associated with the physical location to which the virtual page is mapped. If an attacker VM A wants to corrupt the data written by the victim VM V, it will have to create a page aliasing first. In other words it will have to map one of its pages to the same memory location that was mapped to the page of the victim VM. Provided that the attacker VM A is able to correctly locate the virtual address which corresponds to the physical address of the page of the victim VM V, upon the mapping of a new page of the attacker VM A into a physical location used by the victim VM V, the AMD SP will inspect the Reverse Map Table corresponding to that physical location. The RMP will contain the identifier of the victim VM V, thus it will not allow the attacker VM A to map the page, returning an error. By forbidding the memory aliasing, this technology effectively prevents memory corruption. If instead the attack is performed from a DMA-enabled peripheral, e.g., through a misconfiguration of the Input Output Memory Management Unit (IOMMU) that allows the peripheral to modify a page which is outside the memory buffers dedicated to the DMA transfers, the RMP access control will not be able to stop the attack, but the RMP integrity control will. In fact, when the corrupted memory page is read back by the victim VM V, the integrity check will fail, and an error will be raised.

2.1.7 A Consideration on Trust Foundations

Both Intel SGX/TME/MKTME and AMD SME/SEV/SNP manage encryption keys in an entity which is deeply embedded in the CPU silicon, more privileged than the hypervisor, which itself is at the bottom (ring -2) of the traditional protection rings classification [30, 83]. In fact, Intel ME and AMD SP management engines can be considered as ring -3. Both Intel ME and AMD SP are privileged entities which enforce the security properties of the aforementioned security architectures. The fact that they are both running proprietary undocumented firmware is worrying, in fact no public security audit has been performed on the code running on both systems, which means that vulnerabilities, or worse, backdoors, might exist on both systems. In the past several vulnerabilities have been found in such management systems [32, 73, 101, 119, 120], however this has not led to an increase in transparency by the hardware manufacturers. If through a vulnerability or a backdoor, access to a security system like SP or ME was granted, an attacker could bypass all the security architectures implemented on a system at once. This is possible because both systems have unfettered access to the system memory and hold the encryption keys of all the separately encrypted regions, moreover both systems have access to the network, which for example in Intel systems is granted through an SMBus connection to an Ethernet or Wi-Fi adapter, as required on Active Management Technology (AMT) enabled motherboards and platforms. The publication of the source code of the RTOS that are executed in Intel ME and AMD SP could enable the global security research community to perform a security audit to these platforms, a reproducible build system could allow to attest that the firmware that is in execution in such platforms is authentic and unmodified, only these two actions allow the next step ahead towards trusting these powerful technologies.

2.1.8 The Need For Plaintext Data

All the aforementioned security architectures perform an End-to-End (E2E) encryption, data is encrypted on the host, sent encrypted on the bus, stored encrypted on the memory, eventually retrieved, and decrypted only when it arrives back on the host. E2E encryption has the advantage of being simple to implement, and of keeping the trust boundary limited to the host itself. However there are cases where a system architect might want the memory to store a cleartext version of the data. Emerging memory such as Memristors [124], Spin-Transfer Torque (STT-RAM) [84], ULTRARAM [91], Phase Change Memory (PCM) [128] and 3D XPoint (3DXP) [106], create such a necessity as a consequence of their physical characteristics. In fact emerging memory technologies have a limited endurance which is typically half-way between the DRAM endurance of millions of write cycles and the NAND FLASH endurance of thousands of write-erase cycles [59]. The limited lifetime of such memories pushes for optimized write strategies which are able to reduce the number of writes employing a differential approach. Differential write strategies, during a write operation, enable the memory to read the data which is already stored in the destination, and compare it with the data that will be written into that destination, then an efficient write strategy could only write the bits whose value will change when switching to the new content. However, secure encryption algorithms are characterized by the diffusion property, which ensure that even a single bit change in the plaintext results on average in half of the bits of the ciphertext block changing value [136]. As a consequence, storing encrypted data in an emerging memory renders all the differential write strategies that might be implemented ineffective, eliminating the medium lifetime improvement that they bring.

Another condition that requires the peripheral to have access to cleartext data is the adoption of Processing In Memory (PIM) [65] and Near Memory Processing (NMP) techniques [108]. Such techniques allow to offload a part of the computations on the memory peripheral, such that data movement is minimized. Reducing the movement of data reduces the latency of the computation as well as the power consumption of the system, making the whole computing architecture more performant and energy efficient. Homomorphic encryption schemes [53, 99] might allow to perform computation on ciphertext data, but current implementations have huge performance impact and consequently power consumption overheads, that will nullify the PIM/NMP value proposition. The only chance for the secure implementation of efficient systems is to encrypt at link level, giving the peripheral access to plaintext data.

A link encryption mechanism can only be considered secure if the other endpoint is able to properly secure the plaintext data that will receive and if each endpoint is able to perform a security attestation on the other endpoint. If Perfect Forward Secrecy (PFS) [114] is desired, in other words, if the confidentiality of data should not rely on a single secret or pair of secrets, ephemeral keys should be employed. However, to be able to securely agree on a sequence of ephemeral keys, the two host must perform an attestation process which is based either on symmetric or asymmetric cryptography, which should be used to authenticate a key agreement protocol like Diffie Hellman, to effectively thwart Man-In-The-Middle (MITM) attacks.

2.1.9 Link Encryption

Modern link encryption over digital protocols was born in August 1986 to secure internet communications, a joint project of the National Security Agency (NSA), the National Bureau of Standards, the Defense Communications Agency, and computing equipment manufacturers, led to the inception of the Secure Data Network System (SDNS) [109]. Several years of re-design, extension and deprecation turned that original project into the Transport Layer Security (TLS) v.1.3 protocol that we use every day to secure traffic sent over the internet. Due to his backward compatibility requirements and the broad set of ciphers (CipherSuites) employed by TLS v1.3 [55], the protocol has now reached a complexity that makes it hard to implement in a functional and secure manner. As a further confirmation, only a dozen TLS implementation exist and only a handful of them implement TLS v1.3 [56]. Even though TLS itself might not be directly portable to scenarios different from the World Wide Web, many of the principles used in the TLS protocol have made their way onto modern link encryption implementations. Among such principles are: endpoints mutual authentication, ephemeral keys exchange, use of lightweight stream-oriented ciphers and mode of operation with strict ordering, session establishment integrity guaranteed through hashing of exchanged messages.

There are three notable examples of modern link encryption implementations: MACsec [111], PCIe IDE [117] and CXL IDE [137]. These three protocols, despite being applied to different contexts, like Ethernet networks (MACsec), DMA protocols (PCIe IDE) and cache coherent CPU interconnections (CXL IDE), share a many features. All the three protocols are based on AES-GCM [102], a recent stream-oriented AES mode of operation, which can guarantee confidentiality and integrity of the transmitted data at once, thanks to the computation of a Message Authentication Code (MAC) tag embedded in the encryption and decryption processes. The MAC obtained during the encryption is sent in cleartext along with the message and needs to match with the MAC produced during the decryption of the message. If the two match, the transferred data has not been tampered with. In all the three protocols, the AES-GCM streams forbid the reordering of the blocks, thus hampering all the protocol ciphertext reordering and splicing attacks. This has some consequences, in fact all the three protocols need to reorder some of their packets, e.g., in PCIe to avoid deadlocks, posted requests need can be moved ahead of non-posted requests [117]. This is solved by creating separate encryption streams, e.g., one for posted requests and another one for non-posted requests. In fact blocks belonging to different encryption streams can be reordered freely. MACsec, PCIe IDE and CXL IDE represent solid ground on which future secure protocols, that satisfy unforeseen security requirement of the emerging technologies can be built.

2.2 Compute Express Link Security in Multi-tenant Scenarios

Until now our threat model considered the whole host, together with all the software running on it, as a trusted and secure entity, and all the threats were hypothesized to come from external actors with physical access to the system. However such a threat model might fall short in representing the possible threats of modern data center environments. In fact, in nowadays data centers that offer public cloud services, several virtual machines, belonging to mutually untrusted users, might coexist on the same host, sharing the same CPU, caches, memory subsystem, and DRAM. In the work in progress journal embedded as annex, we will adopt a threat model which is adequate to represent the challenges of such an environment. This new threat model defines three entities: the cloud-service provider, which has privileged access on the VMM and physical access on the whole hardware, the tenant, which has privileged access on one or more virtual machines, and the attacker, which can be also a tenant, and thus can have privileged access on its own virtual machine or even gain physical access on the platform by penetrating the data center in person. The attacker role could also overlap with the cloud-service provider, so we should expect the attacker to have complete control over the Hypervisor. Such a threat model is also aligned to the ones adopted by the QEMU Security Process [148], by AMD SEV/SNP [4, 5] and by Intel TDX [74]. In this threat model we want each tenant to preserve the confidentiality and integrity and to achieve replay protection on all the data-at-rest and data-in-use which is owned by the Virtual Machines under its control.

2.2.1 Multi-tenant Attack Scenarios

Three attack scenarios can be examined, which adhere to the aforementioned threat model: cross-TVM access, physical device access and Rowhammer. We call *Victim TVM* the Trusted Virtual Machine whose data confidentiality and integrity should be protected. Replay attacks themselves can be considered as a threat to integrity, as their net effect is to alter the memory content without necessarily breaking confidentiality. The starting condition of the three attack scenarios is that the *victim TVM* writes a memory page in an arbitrary location which belongs to its address space, and writes arbitrary data therein. The aforementioned security properties will be respected if the following read request, issued by the *victim TVM* to the previous location, will actually read the same data that was last written by the same *victim TVM*. The first attack

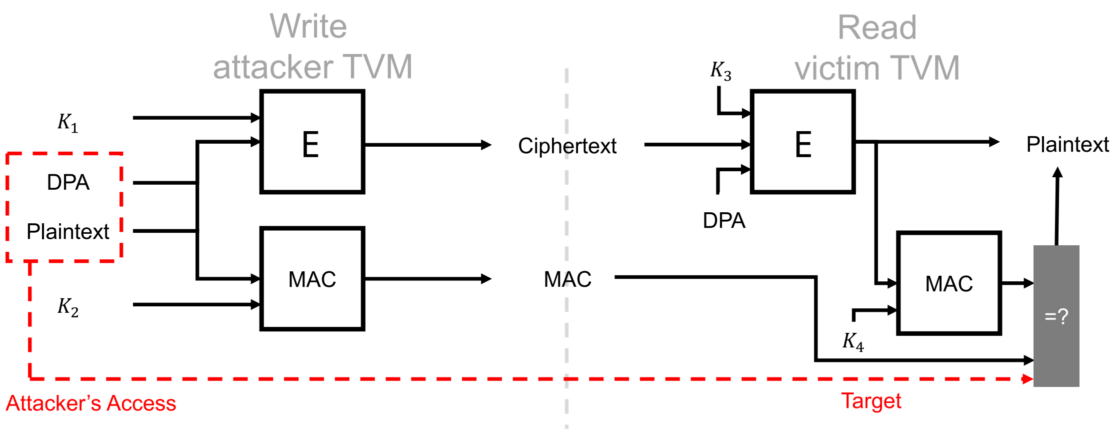


Figure 2.1: *Graphical depiction of an internal cross-TVM attack scenario, considering an encryption and MAC security architecture, the Attacker TVM on the left can manipulate the data and addresses arbitrarily, and it tries to generate a collision on the MAC while it is being read by the Victim TVM*

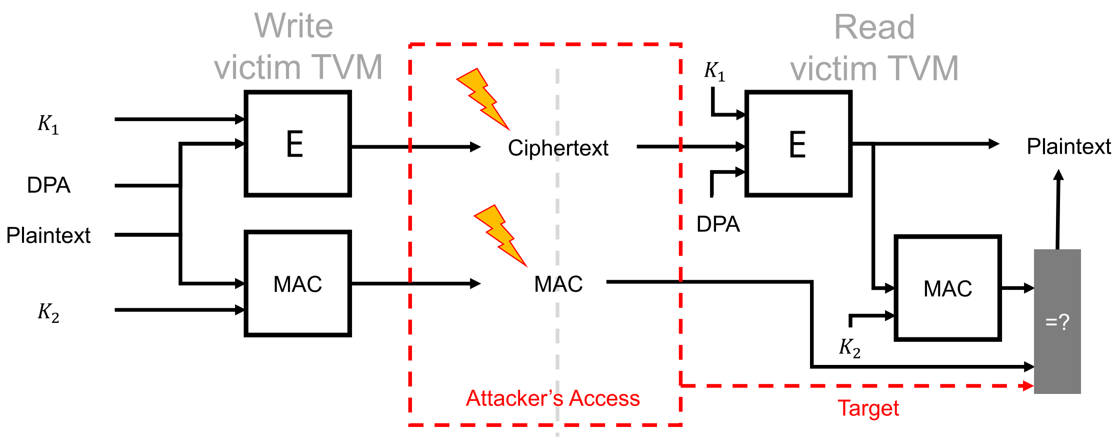


Figure 2.2: *Depiction of an external physical attack scenario on an encryption and MAC security architecture, the attacker on the left can manipulate the data and addresses arbitrarily either while they are stored on the medium or while they are on transit through the memory protocol. The purpose of the attack is again to generate a collision on the MAC being read by the Victim TVM*

scenario, represented in Fig. 2.1, considers an *attacker TVM* to be co-existing on the same VMM as the *Victim TVM*. While usually different TVMs or VMs insist on disjoint memory ranges, there could be aliasing between their address spaces, e.g. in case of shared access to shared DMA enabled memory, configuration errors, or the presence of malicious tenants. In these cases the Attacker TVM is able to issue a write request to the memory location initially written by the Victim TVM, thus overwriting data. A secure memory architecture should either prevent the Attacker TVM from writing to the memory location, or, in case an implementation trade-off cannot afford to guarantee the integrity during memory writes, raise an exception when the Victim TVM reads back the altered data, preventing the silent loading of corrupted data. The second proposed attack scenario, shown in Fig. 2.2, considers the attacker to be able to physically access either the storage medium, that can be composed of traditional DRAM, emerging memory or hybrid solutions comprising both technology; or the physical memory bus,

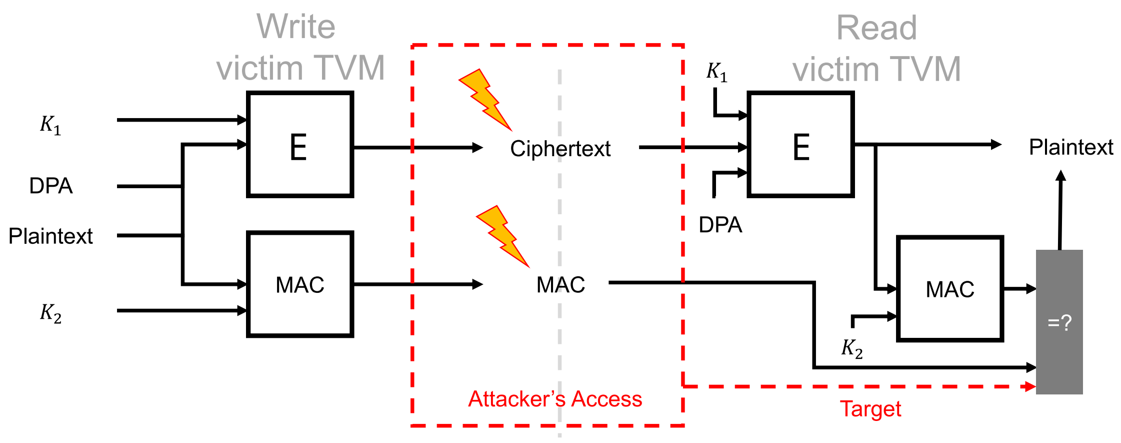


Figure 2.3: *This figure represent the Rowhammer attack scenario in a multi-tenant environment employing a MAC-based security architecture, the attacker on the left issues a memory read pattern with the purpose of causing Rowhammer-derived memory corruption on the medium. The goal of the attacker is to generate a collision on the MAC read by the Victim TVM*

be it a traditional DDR bus [79], a PCI express [116] link, or a CXL [137] link. In this scenario, the attacker physically alters the data stored on the medium or travelling through the protocol, e.g. through a Machine-In-The-Middle (MITM) [18] approach. In this case, the desired level of security is guaranteed, where applicable such as in PCIe or CXL, by cryptographic protection of the protocol with authenticating encryption algorithms such as AES-GCM, or by checking the ownership of the last written data during the readout, in case protocol encryption and integrity checking is not feasible. The combination of both approaches can guarantee the previously mentioned security properties. A third attack scenario, depicted in Fig. 2.3, leverages the Rowhammer phenomenon [123]. A Rowhammer attack exploits the physical characteristics of DRAM memory, which stores data as the charge of capacitors called *cells*. On top of each cell an *access transitor* is placed, which activates (opens) or deactivates (closes) to preserve the cell charge when there is no need to read it. The cells access transistors are activated in lines calles *rows*. *Rows* are grouped into banks, each bank features a single set of *sense amplifiers* which are used to readout the charge stored in each cell. The unified access to the sense amplifiers mandates to have at most one open row at a time. By repeatedly opening and closing two rows belonging to the same bank, an attacker can speed up the naturally occurring leakage of the DRAM cells, rendering it faster than the DRAM periodic refresh action, causing a corruption of the data stored in the DRAM [85]. Provided that an attacker is able to correctly infer the complete address mapping from *virtual TVM* addresses to geometrical DRAM location, which can be inferred using a software-only methodology formalized by Barenghi et. al. [22], an attacker can leverage the Rowhammer phenomenon to corrupt the victim TVM memory by just issuing memory read operations. A particularly dangerous trait of Rowhammer is that the attacker might not even have read access to the corrupted memory location. That is because subsequent accesses to the same bank determine a *bank conflict* and are avoided by the system by applying *bank interleaving* techniques. Thus a Rowhammer attack does not need a misconfiguration or a memory aliasing as a prerequisite. A secure multi-tenant virtualization architecture allows to protect against Rowhammer by either implementing MAC-based integrity control, which would require the Rowhammer corruption to cause a tag conflict not to be detected, or a Rowhammer-aware virtual to physical memory mapping that places entities belonging to the different tenants into geometrically separate memory regions.

2.2.2 Commercial Solutions for Multi-tenant Security

Two commercial solutions are available to achieve trustworthy access to memory in a multi-tenant solution, Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP) by AMD [4] and Trust Domain eXtensions (TDX) by intel [74]. While traditional virtualized setups does not satisfy the properties mentioned in this threat model, AMD SEV and Intel TDX instead allow to guarantee such properties in some attack scenarios. Both commercial solutions leverage the existence of a portion of the system which is under the exclusive control of the hardware manufacturer. That portion of the hardware is the Management Engine in Intel platforms and the Secure Processor in AMD platforms. AMD SEV-SNP [4] maintains a *Reverse Mapping Table* (RMT) in DRAM, a data structure that for each virtual page keeps track of the owner entity and the physical address mapped to that page, forbidding the creation of memory aliasing. The Reverse Mapping Table is checked by the CPU and IOMMU during their page table walks and updated during memory allocation requests. Standard operating systems, including the Hypervisor, cannot directly alter the RMT, thus privilege separation between Hypervisor and TVM can be implemented. In fact if an *Attacker TVM* tries to set up a memory aliasing with a *Victim VM*, the RMP will detect that the physical page was already mapped to a virtual page belonging to a different entity (the *Victim TVM*) and thus forbid the allocation of the *Attacker TVM* page. AMD SEV-SNP currently does not offer protection against physical access to the medium, nor protection against Rowhammer attacks. Intel TDX [74] instead computes a 28-bit MAC digest using a modified Carter-Wegman scheme [46] which is stored on the DRAM medium alongside the data. The MAC digest is computed over the address, data and an identifier Memory Key IDentifier (MKID) of the entity owning the memory page. The digest is then checked only during memory reads, so even though memory corruption itself is not prevented, the consumption of corrupted memory by the *Victim TVM* is prevented by checking the MAC during memory reads. Data stored on the memory medium is also encrypted with a key selected by the TVM identifier, so if an *Attacker TVM* tries to readout the memory belonging to a *Victim TVM*, the memory encrypted with the key associated with the MKID of the *Victim TVM* will be decrypted with the key associated to the MKID of the *Attacker TVM*, thus will not allow the content readout. If the *Attacker TVM* tries to corrupt a memory page belonging to a *Victim TVM*, it will lead to a MAC failure as the MAC will be updated with the key associated with the *Attacker TVM*. However there is a large caveat in both solutions, AMD SEV and Intel TDX do not currently allow to encrypt memory pages which are used for DMA-based communication with external peripherals. This choice is for backward compatibility with unmodified peripherals. In fact a DMA-enabled peripheral, when reading through an encrypted page would not be able to interpret the data stored therein. This limitation implies that the benefits of CXL-based memory expanders and accelerators are entirely inaccessible to users that wish to guarantee the security of their data.

2.3 Mobile Systems Secure State Management

Mobile devices are attractive targets from a Computer Security standpoint for several reasons: they are used daily by more than 6.6 billion of users [17] to receive, produce and transmit sensitive data. Furthermore smartphones are now widely adopted as a mean of payment, second factor authentication for critical services such as digital identity, home banking and tax management, either through SMS One Time Password (OTP) or Time-based One Time Password (TOTP) [48]. The portable nature of mobile devices makes them also easily subject to loss or theft, thus increasing the likelihood of a threat scenario where the attacker has physical access to the device. Even though opening a mobile device and performing hardware modifications requires some equipment such as an SMD rework station, such equipment is readily available at low cost or can be lent from one of the many phone repair shops. In fact, repair shops often replace faulty NAND storage chips from mobile phones, de-facto enabling sophisticated and invasive attacks like the probing and tampering of the data lines between SoC and DRAM and between SoC and NAND.

One of the major discriminant between mobile devices is whether they run iOS (or its variants iPadOS, tvOS, watchOS) or Android (or its variants Android TV, Wear OS). iOS runs on all and only the mobile devices manufactured by Apple, while Android, originally developed by Google is released in many customized forms by thousands of device manufacturers globally, making it the world's most widespread operating system [12]. The two ecosystems are profoundly different, with Apple mobile devices being tightly controlled from the hardware design up to the software that run on the devices themselves. Android devices, instead, are independently developed and diverse from a hardware perspective, but OEMs implement or configure many of the security features as well, leading to a fragmented scenario. In this case a lack of centralization can lead to the delegation of the development of critical features to device manufacturers, which in some cases lack the resources for a proper security auditing of their technologies.

Smartphones and tablets store their own operating system and user files in a device called *managed NAND*. A managed NAND includes one or more NAND banks, as well as a memory controller, able to abstract from the NAND medium and expose a block device interface, accessible through the embedded Multimedia Memory Card (eMMC) [81] or Universal Flash Storage (UFS) [82] protocols. In general, storage read and write commands are not authenticated, therefore any agent able to communicate with the storage protocol (eMMC or UFS) and issue commands can read and write arbitrarily inside the storage of the mobile device. However, to enable the issuing of a security architecture for a mobile device, there is the need to set aside some storage space with confidentiality and integrity properties, which might be enforced with an access control mechanism. To satisfy this need, the two main mobile storage protocols: eMMC and UFS, embed a feature called Replay Protected Memory Block (RPMB).

An RPMB is a Logical Unit (LU) of a managed NAND, on which the controller enforces a set of security properties: any entity wishing to write to such Logical Unit must demonstrate to the controller the knowledge of a pre-shared secret. The knowledge is proven by exhibiting a Message Authentication Code computed over each write command, keyed with the pre-shared key. Read requests, instead, are not protected.

Another security property guaranteed by the RPMB is the write counter. Such counter is incremented each time a write request is issued, and never overflows. This last security feature can effectively thwart protocol replay attacks, where an attacker that does not have knowledge of the pre-shared key is able to listen to valid transactions and replay them, e.g., to restore a security counter to a previous value, or to restore an old firmware digest, enabling the roll-back to an obsolete and vulnerable firmware image. Such mechanisms however are enforced only by the firmware running on the managed NAND controller, therefore inherit the same security level as the firmware itself. It should be noted that managed nand firmware is closed source and executed on a proprietary architecture, furthermore there is no public evidence of any security audit being performed on such firmware. Finally, there is public evidence of a backdoor being implemented in an eMMC firmware produced by Samsung [19, 26, 112], which further discourages from relying on the RPMB as the sole foundation of any security architecture. To improve the performance and reliability of their own mobile devices, manufacturers often execute Field Firmware Upgrades (FFU) of their managed NAND devices. The mobile device receives a new firmware image for the eMMC or UFS and sends it to the managed NAND through a standardized FFU commands, either during the early stages of the Linux kernel boot (Samsung Galaxy S5), during the init process (Samsung Galaxy S9) or during the UEFI environment (Google Pixel 2/3). The consequence of this operation is that firmware images for most of the managed NAND devices can be effectively recovered by dissecting the firmware updates of the target devices which are equipped with them. This further lowers the attack difficulty, enabling an evildoer to easily obtain a firmware image that can be reverse engineered, hunting for vulnerabilities that can invalidate the security properties of the Replay Protected Memory Block. Finally, many OEMs use the RPMB for storing information about the security state of a device, including the locked or unlocked state of the bootloader, the Engineering Sample Device flag, indicating if a device is an engineering sample and has to disable all the security features.

Some manufacturers however follow a different strategy to maintain integrity protected data on the device. In particular, Motorola chose to leverage a set of electronic Fuse (eFuse) that Qualcomm embeds in their System on Chips, under the commercial name *qFuse*. eFuses, patented by IBM in 1989 [27], are portions of an Application Specific Integrated Circuit (ASIC) which inherit the functioning of traditional electrical fuses: by default they act as closed circuits, but if enough current is made flow through them, they heat up and blow, turning into open circuits. eFuses can be used to store a small quantity of information, due to their large size compared to other ASIC storage technologies like NOR or NAND. Furthermore, eFuses have the desirable feature of being one-time writable, as there is no practical way to turn a burned fuse back into pristine state. Even though being one-time writable, eFuses are not by their own nature integrity protected, in fact individual bits of the eFuse bank which are still not blown can be individually blown, therefore for some application like anti-rollback counter, they might be used as they are, but for other application like the storage of public keys, access to the eFuse block needs to be filtered and granted only to privileged entities, otherwise evildoers could alter the public key, which in the case of RSA could make it weaker and easily factorable. Only few manufacturers use store their secrets into eFuse banks, supposedly because their usage limits the possibilities for device refurbishing.

In fact if all the state of a device is stored in the managed NAND, just replacing the managed NAND with an empty one can bring back the device into a clean state, while for refurbishing a device which adopts eFuses, the entire SoC needs to be replaced, which is typically more costly than the managed NAND.

With the increase in popularity of the smartphone-based mobile payment systems such as Android Pay and Apple Pay, and the increasing adoption of embedded Subscriber Identity Module (eSIM) by mobile carriers, the need for a proper Secure Element (SE) to be embedded in smartphones grew. Apple introduced its Secure Element, called Secure Enclave in 2013 with the iPhone 5s. Google introduced its Secure Element called Titan M in 2019 in its Google Pixel 3 (2018) flagship device and Samsung added its own SE in the Galaxy S20 in 2020. While an embedded Secure Element can greatly reduce the attack surface, and thus provide a good improvement in security with respect to other solution presented before, conversely, Secure Elements are currently included only by Apple, Google and Samsung, and only in their flagship smartphones due their additional cost. Google is trying to obviate to this problem by sponsoring the Open Titan initiative [97], which aims to be an Open Hardware, publicly developed and scrutinized Secure Element, but the project is still young and it will first be integrated in server environments and hopefully in a few years will reach the mobile ecosystem. Furthermore, the adoption of a Secure Element does not exclude a priori the existence of vulnerabilities and security flaws, as Bellom et. al. demonstrate in their latest work 2021: A Titan M Odyssey [29], also a Secure Element, especially one which is physically separate from the SoC like the Google Titan M, can contain security vulnerabilities, able to undermine completely the security model of a smartphone.

2.4 Exploring Cortex-M Microarchitectural Side Channel Information Leakage

The existence of side channels, defined as unintentional radiation of information is probably known since World War II era, when the British discovered an effect called *Rafter*, which described the inadvertent radiation of oscillators and receivers, which at the time could have been used as a beacon for homing enemy submarines, as described by Peter's Wright book *Spycatcher* [153]. One of the first historical example in which a side-channel information was effectively used was in the cold-war era, when the US intelligence built the Berlin tunnel wiretap [36] to record encrypted Russian teletype communications, and the analysis of the recorded signal carriers was interrupted by clicks which were caused by electromagnets of the printing side of the teletype, and thus were correlated with the plaintext being printed [51]. The phenomenon was then codenamed as TEMPEST by NATO and NSA, referring to both techniques to extract information from radiated side channels and techniques to shield against such information leakage. However the first public research on the TEMPEST phenomena was published only in 1985 by Wim van Eck [150], which demonstrated the readout of data displayed on a computer monitor from hundreds of meters using only 15$ of equipment. Markus G. Kuhn developed a different low-cost techniques to eavesdrop the data being visualized on CRT and LCD based monitors, and he also developed some techniques to mitigate the leaked signal [88].

2.4.1 Cryptographic Side Channel Attacks

While TEMPEST allows the direct retrieval of plaintext, if a side channel attack is applied to a cipher instead, the effort to retrieve a small key can lead to the successful decryption of a large amount of plaintext. Furthermore, a cryptographic key is usually processed iteratively several times, thus giving the attacker a large quantity of samples which are correlated with the key material, greatly simplifying the attack. The first modern cryptographic side channel attack was described by Paul C. Kocher, in 1996, and introduced the idea of observing the variation of execution time in different cryptosystem to infer the key, at the same time presenting a set of complete key recovery attacks [86]. After that pioneering work, many other cryptographic side channels were explored, such as power consumption [87], Electro-Magnetic (EM) emissions [61], optical emissions [58] and acoustic emissions [20]. The EM emission side channel was chosen for our work since the necessary instrumentation is affordable and, conversely to the optical side channel, does not require decapsulation (also called decapping) of the Integrated Circuit (IC). A typical EM side channel setup is composed by an antenna placed close to the device under attack, as depicted in Figure 2.4, which is connected to an oscilloscope to measure the EM leakage, and to a computer able to collect traces and control the device under attack. Thus in general cryptographic side channel attacks can only be performed from a close distance; a notable exception is a work by Camurati et. al., which has shown that in devices in which analog and digital logic are implemented in the same die, the EM leakage from cryptographic operation of the devices can be retrieved from the noise of the signal emitted by the analog portion, thus enabling a Side Channel attack from a distance of several meters [34].

2.4.2 Attack Classes

The first side channel attack proposed by Kocher in 1996 [86], exploited the correlation between data and execution time in RSA computations. This kind of side channel attacks, where the information is already visible through a visual inspection of the trace is defined Simple Power Analysis (SPA). Additional variants of that attack might be applicable to software implementations, in which microarchitectural features which are designed for performance might leak information about the execution trace. As highlighted by Lou et al. [96], an instruction cache will have faster access time on the instructions which were already executed, thus if the control flow is dependent on the input data, the execution time can be correlated with the data itself and thus be exploited to retrieve the secrets being computed.

A more powerful technique was proposed by Kocher in 1999 [87] called Differential Power Analysis (DPA), this technique is more flexible than SPA since it does not require the attacker to possess a detailed knowledge about the cipher, in particular it is not a prerequisite the knowledge of the timing at which the signal was correlated with the secret, the only requirement is to have correlation between the signal and the secret and to have that correlation repeating over time. If an attacker finds an intermediate value which is correlated only with the plaintext (or ciphertext or both) and a small portion of the key, she can collect a large number of execution traces for different plaintexts, group the traces according to the value of a portion of the intermediate, and look at the average of the two groups. Since the leakage is correlated with the intermediate values,

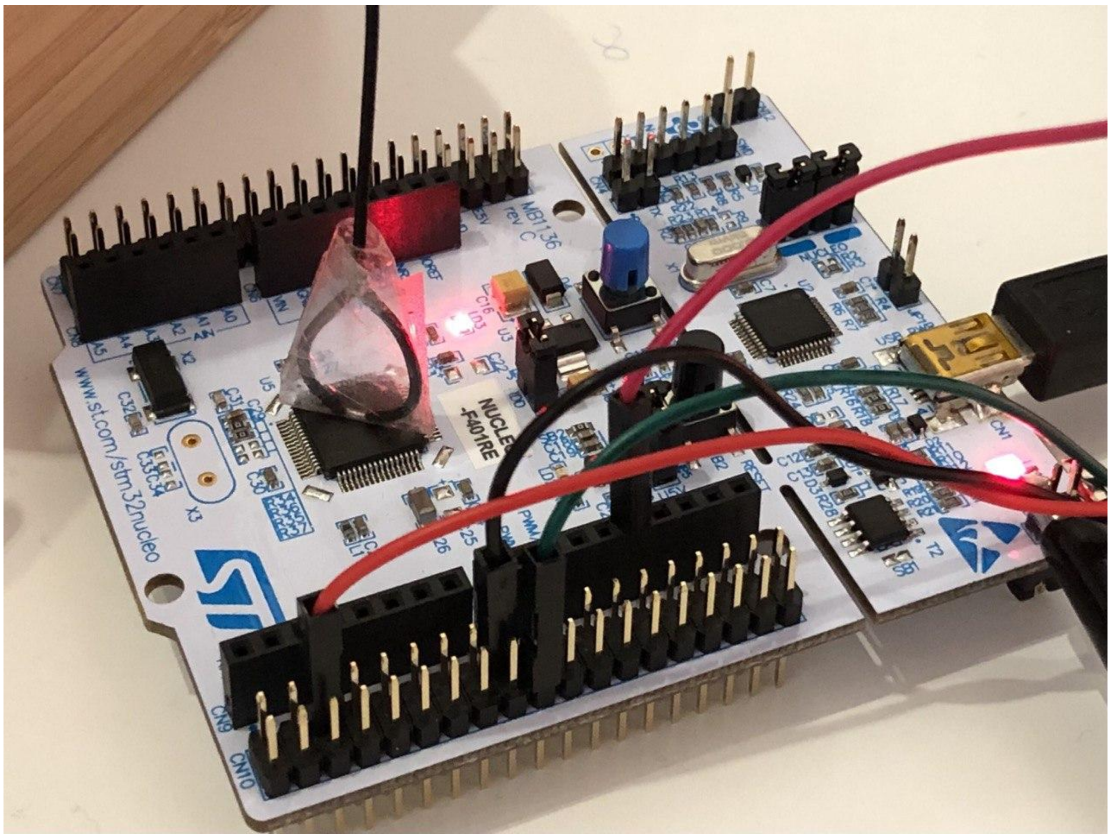


Figure 2.4: *Depiction of a portion of the Side Channel setup used to perform EM measurements on the Cortex-M4 CPU in our second work. A self-built electric field antenna is put on top of the Micro Controller Unit (MCU) package containing the Cortex-M4 processor. The other wires in the picture are used for a GPIO-driven trigger to align traces and an a serial port for controlling the software in execution on the CPU.*

the key portion that makes the groups differ sensibly will likely match with the correct key.

A more refined version of DPA is called Correlation Power Analysis (CPA) and was formalized by Brier et al. in 2004 [31]. CPA works by ranking all the hypotheses for a portion of the key, e.g., a single byte, considering a large set of traces acquired during the encryption of different known plaintexts, according to the statistical correlation of the intermediate values generated by each plaintext and that key hypothesis. The Pearson's correlation coefficient is a statistical estimator that is able to quantify the linear correlation between two datasets, and is used in this attack to estimate the correlation between the intermediate values computed with a key hypothesis and the corresponding traces. Furthermore, Heuser et. al. demonstrated the optimality of the Pearson's correlation coefficient distinguisher whenever the leakage model is only known to a proportional scale (i.e. $L_{\theta,d} = ax + b$ where both a and b are unknown), and the noise is Gaussian [70].

The last major class of side channel attacks are called template attacks and are the most powerful attacks from an information theoretic perspective. Template attacks were introduced by Chari et. al. in 2002 [38], and apply a technique employed in telecommunications to retrieve signals in noisy environments, where the signal and the noise are thoroughly characterized to build a model, which is exercised in reception to detect the correct value of each signal. In the same way, a template attack builds a model of the noise over a large number of traces produced by a device which can be completely controlled by the attacker, and is able to retrieve the secret on the target device using a reduced number of traces or even a single trace. One of the latest successful applications of a template attack was carried out by Lomne and Roche in 2021 [131], by successfully extracting an Elliptic Curve Digital Signature Algorithm (ECDSA) private key from a Google Titan Security Key with just 6000 traces. The Google Titan Security Key is a FIDO Universal 2 Factors (U2F) [140] authentication token which is commercially available since 2018 and based on the NXP A700X secure element. However, in our work we will not analyze template attacks, since they are based on the assumption that the attacker can obtain an identical copy of the device to be attacked, and can completely control such device, such that arbitrary input can be fed and used to train the model of the noise. This requirement is far more stringent than CPA, and General Purpose commercial CPUs are still lacking a widespread protection against CPA, thus the simpler attack scenario of CPA better deserves our attention.

2.4.3 Side Channel Countermeasures

Many hardware based side channel countermeasures exist [60, 67] like dual rail [40, 125, 139], triple rail logic [95], and current mode circuits [98], however they are typically not compatible with standard cell libraries and tools commonly used in hardware manufacturing, thus have a high development cost, worse power consumption and performance figures, and are usually only included in high-cost hardware secure enclaves and smart cards, where security is the key priority and the size and complexity of the design is limited. Applying a side channel countermeasure to a General Purpose CPU would probably not be sustainable from a cost perspective, furthermore hardware countermeasure cannot be applied on already existing commercial CPUs, for which the only route is the design of software countermeasures. Software side channel countermeasures can be categorized in three groups according to their working principle. Countermeasures which aim at reducing the SnR between the secrets leakage and the rest of the EM emissions are denominated as hiding [24, 103, 135]. Hiding countermeasures are typically the easiest and cheapest to implement, since they try to modify the implementation, e.g., rescheduling instructions, to minimize the leakage of cryptographic secrets, however for this class of countermeasures only empirical proofs of resistance can be provided and no guarantee is given from the information theory standpoint about the absence of a correlation between secrets and physical variables variation. A second category of countermeasures, named *morphing* tries to mitigate DPA and CPA attacks by modifying the structure of the algorithm implementation thus preventing the statistical aggregation of data. Several examples of morphing countermeasures were proposed [8, 9, 16], e.g., changing the algorithm implementation through dynamic code recompilation [8], masking the values result of load and stores operations [10] or splitting the implementation into a set of tiles, whose order is chosen at runtime by the underlying hardware [16], solution which however requires the augmenting of the hardware platform, thus it's still not applicable to Commercial-Off-The-Shelf (COTS) General Purpose CPUs. Moreover, morphing countermeasures might be vulnerable against

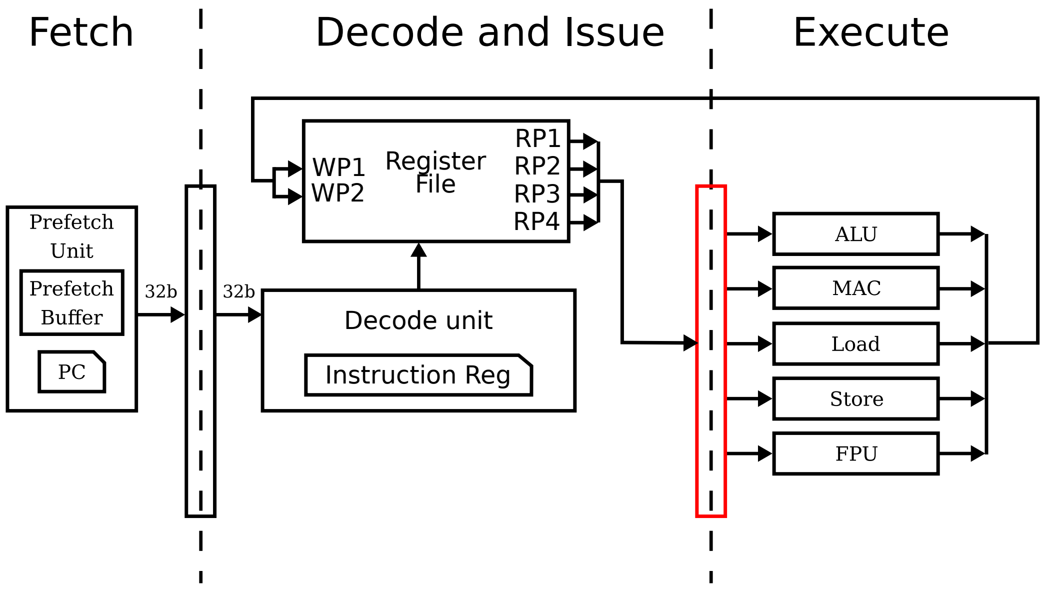


Figure 2.5: *Block diagram of the four stage pipeline employed in the Cortex-M4 CPU. The four stages are the Instruction Fetch (IF), Instruction Decode (ID), ISsue (IS) and EXecute (EX). Highlighted in red is the Issue to Execute (IS/EX) buffer which was proven in the second work of this thesis [23] to be responsible of leaking the Hamming Distance (HD) of the source registers of consecutive instructions.*

template attacks, e.g., if an attacker is able to predict at what time the implementation will embody a particular configuration and can leverage a model of the leakage which is trained for that configuration. Finally, countermeasures which try to break the correlation between the emissions and the ciphertext are called *masking* and they are the only countermeasures able to forbid a side channel attack up to the order of the mitigations. E.g., first order masking can prevent statistical attacks up to the first order but is vulnerable to second order attacks and so on. Intuitively, the number of traces and processing power required to build a higher order attack raises, up to making the attack infeasible or simply not economically convenient. Several examples of masking were proposed in the state of the art [37,63,130]. Masking countermeasures are the only countermeasures for which a formal proof of effectiveness can be derived, provided that a set of invariant properties hold. In fact, if the correlation between the shares of one value is never shared, the masking scheme can be considered secure against statistical methods of the same order of the masking. We will focus on this last category of countermeasures and how to guarantee the validity of their invariants even when implemented in software in a commercially available opaque ISA.

2.4.4 Microarchitectural Characterization

It has been widely demonstrated that hidden architectural effects can undermine the effectiveness of a masking scheme, even if proven secure under from the ISA perspective [21, 43, 52, 63, 104, 113, 135]. As an example let's suppose that a secret value a is split into two shares a_0, a_1 using a random value x, and those shares are then stored then stored into the CPU registers r_0 and r_1.

$$a = (a_0 \oplus a_1), a_0 = (a \oplus x), a_1 = x, r_0 = a_0, r_1 = a_1$$

And the same happes for a secret value b, split using a random value y, and the shares are then stored into registers r_2 and r_3.

$$b = (b_0 \oplus b_1), b_0 = (b \oplus y), b_1 = y, r_2 = b_0, r_3 = b_1$$

A XOR operation securely implemented to operate on the shares, will operate like this:

```
XOR R0, R0, R2
XOR R1, R1, R3
```

If the microarchitecture leaks the hamming distance of the source operands, e.g., from an inter-stage buffer between the ISsue and the EXecute stage (IS/EX), as highlighted in Figure 2.5, the following values will be leaked:

$$r_0 \oplus r_1 = a_0 \oplus a_1 = a$$

$$r_2 \oplus r_3 = b_0 \oplus b_1 = b$$

In fact, they will completely invalidate the masking scheme.

A pioneering work in architectural reverse engineering for the improvement of side channel masking countermeasures was published by Barenghi et. al. [25]. The work enclosed in this thesis stems from that initial effort, standardizing the analisys framework into a formal procedure.

CHAPTER 3

Contributions

In this chapter we will review, for each of the works composing this thesis, the contributions to the state of the art.

3.1 A Secure and Authenticated Host-to-Memory Communication Interface

Emerging memories like Memristors [124], STT-RAM [84], PCM [128] and 3DXP [106], have the key features to change the future computing landscape. In fact they provide a persistent, scalable and byte-addressable medium, with latency and bandwidth figures that are half-way between NAND flash and DRAM. These emerging memories can express their potential in different scenarios. Emerging memories can be interconnected either through storage-oriented protocols like the Non-Volatile Memory express (NVMe) [110] bus, a storage-oriented bus based on PCIe [116] or through main memory-oriented protocols like the traditional Double Data Rate (DDR) class of protocols. One emerging-memory based product which is already commercially available is a 3DXP-based drive connected using an NVMe bus, commercialized under the name Intel Optane [44]. Intel Optane drives can be used as a fast cache for storage devices, technology made available by Intel for the Windows Operating System under the name of Rapid Storage Technology (RST) [45]. Another application of persistent emerging memory is the Direct Access (DAX) technology [90, 152] implemented in the Linux kernel, which offers an Application Programming Interface (API) [118] to bypass the page cache infrastructure and access in a byte-addressable way data directly from the persistent memory, without copying data in the DRAM and caches first.

This paper envisions one possible future integration of an emerging memory device using the standard main memory protocol LPDDR4 [80]. The main contribution in this work is the design and evaluation of a security architecture, able to secure an LPDDR4 link used to communicate with a persistent emerging memory device. The proposed security architecture is able to guarantee confidentiality, integrity and reorder/replay protection on the data transferred over the LPDDR4 bus, as well as integrity and reorder/replay protection on addresses sent along. The proposed architecture aims to minimize the bandwidth and latency overheads, while keeping the silicon area cost affordable, such that the system can be implemented in resource-constrained devices such as mobile or Internet of Things (IoT) devices. To this extent, an authenticated symmetric cipher was chosen for the link encryption, the CAESAR [49] competition finalist ACORN [71] was selected as it's able to achieve a throughput of 9.09Gb/s occupying an area of 9469.8 μm^2, when manifactured using TSMC 65nm technology, according to Kumar et. al. [89]. The cipher was modified to defer the validation of the authentication tag to avoid an increase of latency. A notable achievement was the exploit of the symmetric nature of the ACORN cipher to achieve bidirectional encryption/decryption of the DQ data using a single instance of the cipher. A sequence number and the memory address are used to tweak the key, so that spatial and temporal correlation of transactions can be hidden. To simplify the ecosystem management and facilitate interoperability, the proposed architectures uses a Trust On First Use (TOFU) [151] policy, as the memory peripheral trusts the first host that it sees and accepts a master symmetric secret sent in the clear, since the first connection to a new host is supposed to happen under controlled conditions. The master secret can be erased through a vendor-unique command only after the full erasure of used data, to avoid any information leakage. One innovative feature of the proposed architecture is the possibility to send interrupts to the Operating System in case there is an authentication failure, this can open many possibilities of reactive security policies. For example the Operating System could signal the process whose transaction has failed memory request or generate a higher-level log even that could lead to a faster isolation of the threat and a better damage control, reducing the chance of privilege escalation and data leakage.

Furthermore, this work leaves room for future improvements in several directions, for examples, an encrypted key agreement protocol based on a public-key cryptosystem could be implemented, to apply the TOFU policy without transmitting the master symmetric key in the clear. This work could also see a better clarification of the derivation of ephemeral keys from the master symmetric secret. A potential mapping of the device erase request to vendor-unique LPDDR4 commands could also be a useful extension. Finally, since the tag exchange and validation mechanism is asynchronous, a detailed description of a queue system for outstanding requests that were not yet validated, as well as a back-pressure mechanism to avoid the invalidation of security policies in case of high memory request pressure could also be investigated.

We highlight that the contribution presented in this work is the first publicly described application of a link-level encryption mechanism to an LPDDR4 link, no previous implementation was designed in the academic or industrial world. This protocol pre-dates open link encryption implementations like PCIe IDE [117] and CXL IDE [137], but it already envisioned the fundamental building block of such architectures like the use of authenticated symmetric mode of operations and the implementation of a symmetric-key based authentication mechanism, as well as a secure erase command, able to wipe data and cryptographic keys.

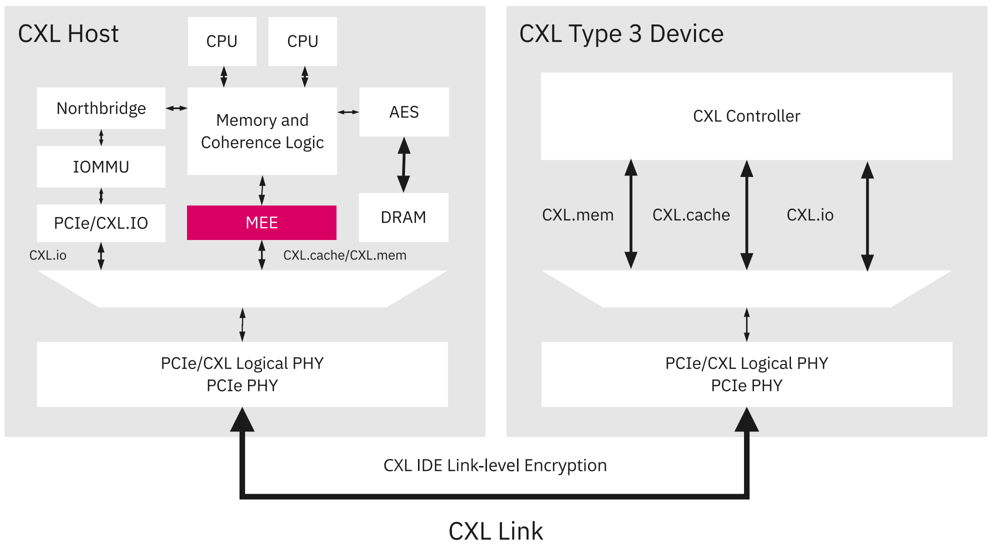


Figure 3.1: *Block diagram representation of the proposed security architecture. The block diagram includes the memory subsystem of a CXL-enabled CPU, attached via CXL bus to a CXL memory expander device. In red are highlighted the functional blocks where memory requests are encrypted to enforce the separation between TVMs and other TVMs or the VMM.*

3.2 Compute Express Link Security in Multi-tenant Scenarios

If one tries to apply the threat model defined by AMD SEV/SNP [4, 5] and Intel TDX [74], to a CXL-enabled platform, one would notice that some threats are effectively protected by existing technologies. For example the CXL IDE [137] performs a link-level encryption which can effectively mitigate the physical threats on the link. Furthermore, for Virtual Machines protected by SEV/SNP/TDX, their DRAM-based main memory is encrypted and not accessible by other entities, as guaranteed by the SEV/SNP/TDX security architectures. However if a VM wishes to leverage the capabilities offered by CXL-enabled peripherals, there are no guarantees on the enforcing of isolation between VMs and between VM and Hypervisor on the data that is sent through CXL links. For example, the Hypervisor retains access to the pages used for Memory Mapped Input Output (MMIO) based configuration of the CXL devices, and such pages are not encrypted by SEV/SNP/TDX, therefore an Hypervisor could always re-configure a CXL device, e.g., setting up an aliased memory map to obtain access to confidential data stored in the CXL device.

CXL is composed of three sub-protocols, multiplexed on the same PCIe PHY: CXL.io for discovery, configuration and non-coherent memory transfers, CXL.mem for cache-coherent memory transfers and CXL.cache for managing cache semantics [137]. A typical CXL-enabled architecture is depicted in Figure 3.1, CXL.io takes the traditional path of non-cache coherent memory transfers, passing through the northbridge which handles all the high-speed peripherals such as PCIe interfaces. CXL.cache and CXL.mem, instead are cache coherence and have low-latency requirements, thus are directly managed by a CXL controller, interconnected directly to the cache subsystem.

A traditional way to give exclusive control over a PCIe periperal to a single Virtual Machine is to use an Input Output Memory Mapping Unit (IOMMU) [13]. An IOMMU is able to translate a dedicated peripheral address space into the host physical address space, performing access control for each memory transaction. To give a VM the exclusive control of a peripheral, the IOMMU can be configured to remap the address space of the peripheral to the physical host address range which is mapped to the address range of the said VM. The VM will detect the peripheral as if it was directly connected. The presence of an IOMMU is identified in AMD systems by the keyword IOMMU and in Intel systems by the commercial name VT-d. However, as highlighted by Malka et. al. [100], an IOMMU is only able to filter upstream traffic, as it is mainly designed to protect against buggy or malicious peripherals, not to isolate VMs between one another or between a VM and its Hypervisor. Therefore the IOMMU alone is not suffient to guarantee the confidentiality, integrity and replay protection on VM data in a multi-tenant threat model.

In this work we propose a secure architecture which bears minimal modifications with respect to the traditional one just described, and is able to extend the properties introduced by SEV/SNP/TDX to the data read and written from a CXL-enabled peripheral, By applying an efficient encryption scheme, the data will be sent encrypted to the CXL peripheral that becomes effectively semi-trusted, that means trusted only regarding the availability of resources. The storage of encrypted data in the CXL peripheral will also prevent cold-boot attack styles that are made simpler by the persistence characteristics of the emerging memory technologies. The encryption key generation and setup is to be performed by the trusted entity which is already present in present-day confidential computing systems such as AMD SEV-SNP [4] and Intel TDX [74]. The proposed solution enables the secure used of CXL peripherals by Trusted Virtual Machines, without having to enlarge the trust boundary, nor to trust the firmware of CXL peripherals.

One possible future improvement of this work is the addition of the Multiple Logical Device (MLD) [137] capability. In fact the CXL standard allows to share a single physical CXL device to different hosts, which can access a portion of the device memory storage through a logical device. However in the MLD management, as defined by the CXL standard, all the Logical Devices (LD) share the same CXL.io memory-mapped address space, therefore an access control cannot be enforced at address level. A potential future solution to this issue implies the use of a technology called Single Root Input Output Virtualization (SR-IOV) [115]. The SR-IOV technology allows to expose multiple PCIe configuration address spaces, which can all use a single physical PCIe interface. Such a technology would provide a diversification in the configuration address spaces for each LD that would enable a direct porting of the security architecture described in this work with a minimal adaptation effort. Even though SR-IOV support is CXL is not already present, the technology was mentioned in the CXL specification [137] as a potential feature, which could be easily ported thanks to the similarities between CXL and PCIe 5.0 in terms of PHY and MMIO mechanisms.

3.3 Mobile Systems Secure State Management

A mobile device security state management architecture, to be applicable in a real-world device scenario, needs to blend security properties as well as manageability properties. We define a list of such properties, based on the observation of the state management architectures of mobile devices which are already on the market. Among those properties are:

a) *Cryptographic unlock procedure:* the OEM must retain the ability to break the device chain of trust, however such operation must be authenticated using cryptographic means, i.e., by exhibiting a cryptographic signature that can be validated with a trusted public key.

b) *Multiple states management:* the device manufacturer must be able to define multiple security states, each one with an independent set of active security features. As an example an engineering sample device should enable the baseband firmware to be replaced, while a device with unlocked bootloader might still require the baseband firmware to be cryptographically valid.

c) *Easy and secure refurbishment:* the refurbishment of the device, typically performed by replacing the managed NAND should remain feasible, e.g. by not relying on a shared secret between SoC and managed NAND.

d) *No trust in storage medium:* the security architecture should preserve its security properties even in the case of a complete compromission of the managed NAND controller security, as it was documented for Samsung eMMC drives [19, 26, 112].

e) *Security downgrade implies full wipe:* every security downgrade should not be feasible unless the user data is wiped, or alternatively, if a security downgrade is performed, user data is immediately and inevitably deleted.

The security architecture which we propose satisfies all of the above properties, however it is also dependent on three hardware prerequisites, which to the best of our knowledge are satisfied by many mobile System on Chips, and mainly by the ones manufactured by Qualcomm, which are the most widespread in the top 20 most popular smartphones, according to the LineageOS public statistics [122]. The hardware prerequisites of the proposed security architecture are:

a) *Trusted Environment:* the logic implementation of the architecture must be executed in an environment which is trusted and separate from the main operation logic, either by executing it in the early boot stages or running it into the TrustZone environment which is present in all Cortex-A CPUs, adopted by the vast majority of smartphones.

b) *One-Time-Programmable Memory:* the mobile device should be equipped with a One-Time-Programmable (OTP) memory, whose access should be restricted only to the software running in the aforementioned Trusted Environment. Qualcomm SoCs are all equipped with eFuses that can act as an OTP. If the Trusted Environment is implemented with TrustZone technology, access to OTP should only be granted to the highest privilege leveles, i.e. the TrustZone kernel.

c) *Replay Counter:* a non-overflowing monotonic counter is needed to thwart replay attacks, where an attacker could use an older valid signature to roll back the target device into a previous state. In case the device is equipped with eFuses, they can be used to implement OTP as well as to implement a replay counter mechanism.

The state storage architecture we propose stores the state of the device into a numerical value called Status IDentifier (SID). The OEM defines in its security policies the mapping between each SID and a selection of security features which must be disabled. The default SID (also assigned in case of error) enabled all the security features. The SID is stored in a certificate signed by the OEM itself, which can be safely stored in the untrusted non-volatile storage, like an eMMC or UFS managed NAND, and is loaded and validated at each boot by the Trusted Environment. The boot process is exemplified in Fig. 3.2.

The secure erasure of user data in case the security level is changed is achieved by storing the Disk Encryption Key (DEK) encrypted with a Key Encryption Key (KEK) that is derived from the SID in the Trusted Environment. If the SID change, the KEK inevitably changes, and the old DEK will not be decryptable anymore, thus will ensure the cryptographic lock of user data. During boot, the Trusted Environment will check the integrity of the SID certificate, and if valid will decrypt the DEK and return it to the bootloader.

The presence of logic fallacies and vulnerabilities is further reduced by ensuring that the dependence between validation of the SID certificate and decryption of the DEK is cryptographically bounded. This can be obtained by deriving the KEK as the result of a keyed hash of the SID and a the monotonic counter c of the device, as well as a device unique key, which is a random per-device secret generated at the first boot of each device and stored in OTP memory, and only accessible by the Trusted Environment.

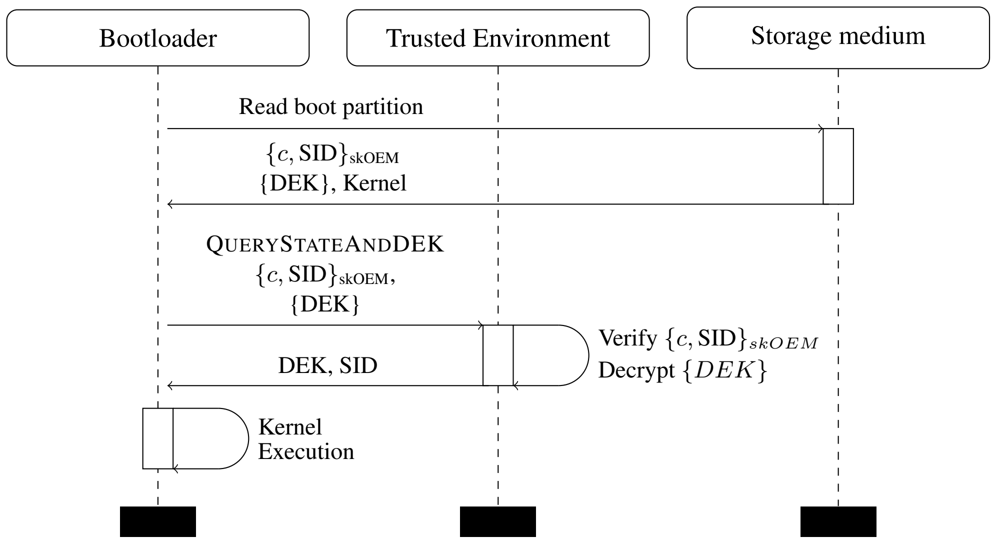

Bootloader
Trusted Environment
Storage medium
Read boot partition
$\{c, \mathrm{SID}\}_{\mathrm{skOEM}}$
$\{\mathrm{DEK}\}$, Kernel
QUERYSTATEANDDEK
$\{c, \mathrm{SID}\}_{\mathrm{skOEM}}$,
$\{\mathrm{DEK}\}$
DEK, SID
Verify $\{c, \mathrm{SID}\}_{skOEM}$
Decrypt $\{DEK\}$
Kernel
Execution

Figure 3.2: *Sequence diagram of the proposed boot process. The resulting SID is trustworthy and is used to toggle security features, such as Verified Boot and dm-verity.*

The SID can be updated, e.g. following a request by the user to unlock its device, only when the OEM sends a new signed certificate with updated sequence number, containing the new SID. The OEM might want to perform additional checks before a

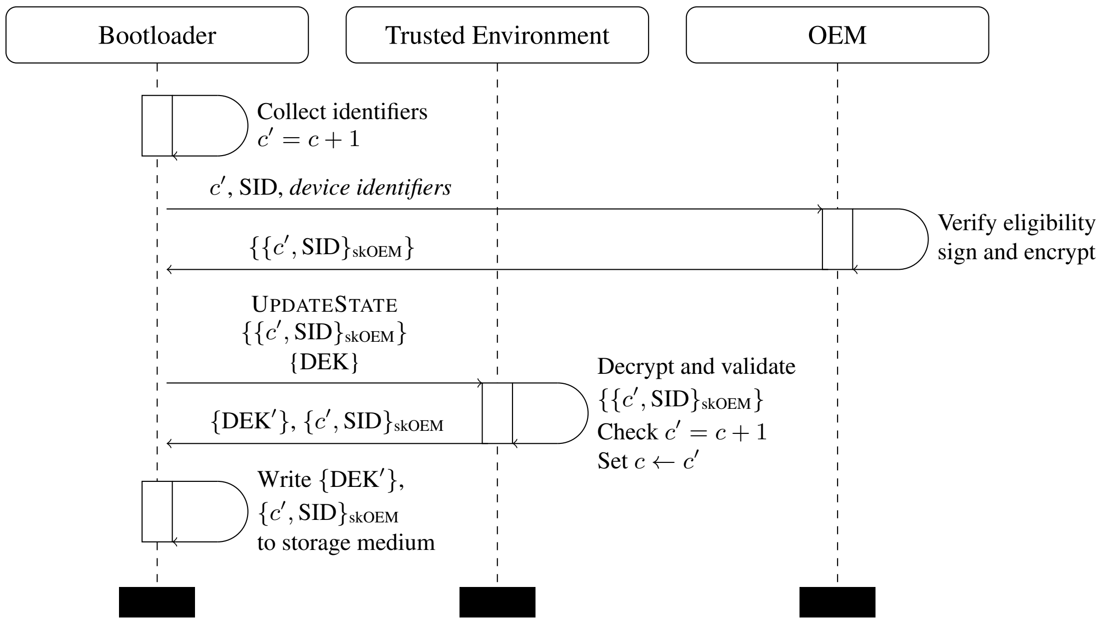

Bootloader
Trusted Environment
OEM
Collect identifiers
$c' = c + 1$
c', SID, *device identifiers*
Verify eligibility
sign and encrypt
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
UPDATESTATE
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
$\{\mathrm{DEK}\}$
Decrypt and validate
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
Check $c' = c + 1$
Set $c \leftarrow c'$
$\{\mathrm{DEK}'\}$, $\{c', \mathrm{SID}\}_{\mathrm{skOEM}}$
Write $\{\mathrm{DEK}'\}$,
$\{c', \mathrm{SID}\}_{\mathrm{skOEM}}$
to storage medium

Figure 3.3: *Sequence diagram of the proposed unlock process. The request incorporates an OEM-specific set of device identifiers, the desired SID, and counter value $c+1$. In case the OEM approves the unlock, the request is signed, encrypted and sent back. The response is then passed to the trusted environment for decryption, ensuring* c *is incremented.*

device could be unlocked, for example, chinese device manufacturers try to avoid third-party re-branding of their own products by requiring a smartphone to be activated with a SIM card and registered with a valid used account and used for a certain amount of time before it can be unlocked. To give flexibility to the OEM to decide when a device is eligible for a SID change, the device sends along with the SID change request also a collection of statistics which could include the total uptime, the IMEI or the presence of a SIM card. If the OEM approved the device security status change it replies to the request with a new signed certificate, including the new SID which is encrypted using a Shared Key (SK) which is accessible only to the Trusted Environment. This SK itself could be burned during manufacturing into the eFuses of the device, and should be differentiated for each device to minimize the risk of security breaches in case one device is compromised and its SK is leaked. The device unlock procedure is depicted in Fig. 3.3.

Each new status change increments a counter c by one, such counter could be kept monotonic and non-overflowing, e.g. by implementing it into an eFuse array using a one-hot encoding. Each time the OEM issues a new security state, the counter c is incremented by one, the device, each time it applies a new security change, will burn one eFuse more, to ensure that the number of blown fuses reflects the counter contained in the certificate. This can avoid replay and rollback attacks where an evildoes can feed an old unlock certificate to a device which was previously unlocked and locked again.

To prove the soundness of the proposed security architecture, a brief analysis of the security properties of the proposed architecture follows, a formal security proof instead is embedded in the attached manuscript. As the Trusted Environment remains available throughout the operation of the device, an attacker could try to obtain the DEK while the device is still locked, perform a backup of the managed NAND, then unlock the device and use the previous DEK to use the data on the unlocked device. We thwart such attack by making the KEK dependent on a monotonic counter, which is incremented every time the SID is updated. This has the net consequence of forbidding any attacker to roll back to a previous state while maintaining the accessibility of data.

The anti-rollback counter c is updated as soon as the new certificate received from the OEM is been validated, attacker might attempt to bypass the counter update by intercepting the new certificate transmission and storing the new certificate in the managed NAND without feeding it to the Trusted Environment. Our architecture effectively prevents this threat by encrypting the certificates which are sent from the OEM to the device with the pre-shared key SK. Certificates are only decrypted during the UPDATESTATE function calls which atomically update the counter c. The described attacks is depicted in Fig. 3.4.

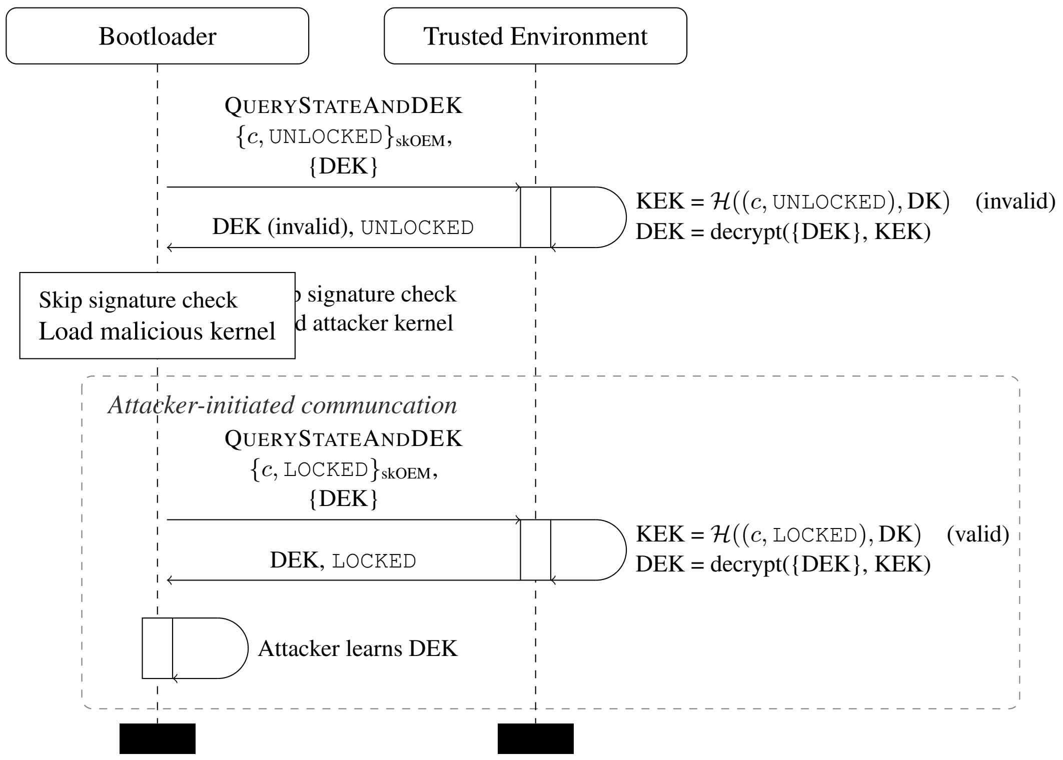


Figure 3.4: *Possible DEK recovery attack in case unencrypted certificates are used.*

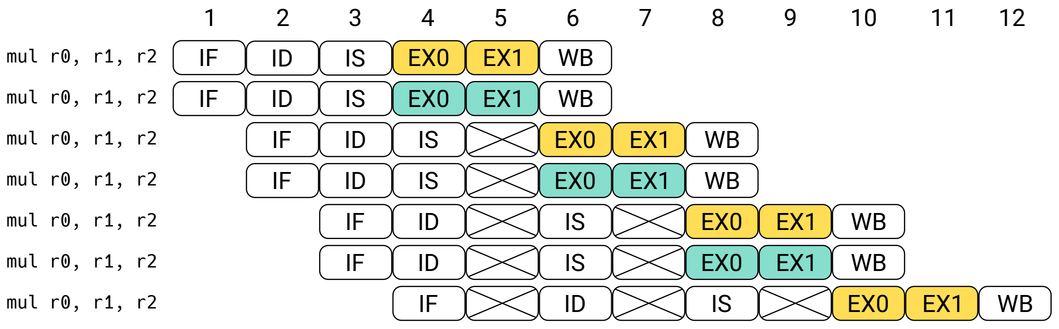
The Trusted Entity as part of our architecture was chosen to be internally stateless, except for the explicit entities declared before like the counter c. This can prevent attacks that rely on a reset of the Trusted Environment, either by exploiting a bug and making the Trusted Entity crash, or by controlling the Power Management Integrated Controller (PMIC) and altering the voltage thresholds of the Trusted Entity, forcing a TE crash and consequently a TE reboot.

3.4 Exploring Cortex-M Microarchitectural Side Channel Information Leakage

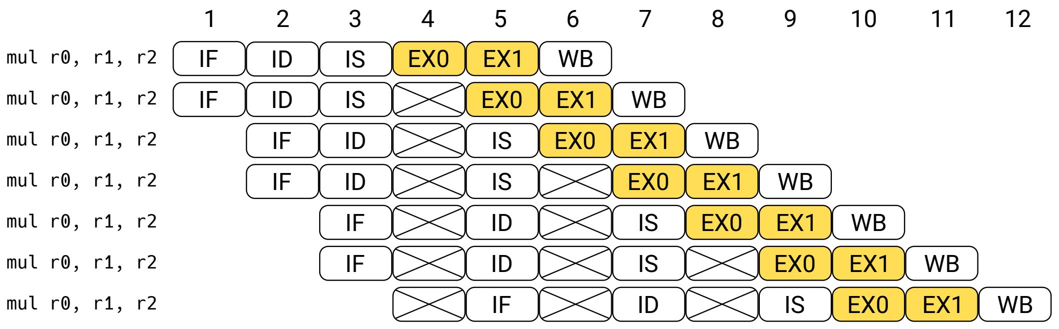
To effectively implement a masking countermeasure on a commercial General Purpose CPU, we need to be able to schedule instructions such that no involuntary recombination of the shares is performed by the CPU microarchitecture. Since information about the re-combination of values below the ISA level is not available for the users and developers of the CPU, there is the need for a reliable and standardized technique to infer that information. Recombination of values is observed on synchronous elements of the datapath, also called registers or buffers, which fulfill different roles inside the CPU. To be able to completely describe the leakage effect of all the synchronous registers inside a CPU, we need to model three structural characteristics of the CPU itself which influence the timing in which the instructions are processed by the functional units of the CPU: 1) data storage features, such as the number of register file read and write ports, 2) instruction issuing features comprising the issue width, the issuing policy, the presence of forwarding paths and the width of the inter-stage registers and 3) instruction execution features that describe the number and type of the available execution units, including their latencies.

General Purpose (GP) Commercial-Off-The-Shelf (COTS) CPUs are designed with performance in mind; they try to execute instructions as fast as possible. For this reason modern CPUs leverage some degree of pipelining, such that the functional units that constitute the stages of execution for each instruction are not left idle between one instruction and the next. However, the code in execution can contain conflicts, or hazards, which might require one or more stall cycles to be inserted in the pipeline to be solved. Architectural optimizations like forwarding paths and replicated functional units can prevent the need for stall cycles. Therefore, by observing the execution throughput as Cycles Per Instruction (CPI) for different sequences of instructions, we will be able to extract information about the presence of optimizations in the datapath, as they will make an instruction sequence with conflicts be executed with a throughput which is as close as possible to the throughput of an instruction sequence without conflicts. An example of a conflict is the Read After Write (RAW), which holds whenever the source register of an instruction is also the destination register of the previous instruction. To avoid the exploration of the whole ISA instruction space, which is impractical, we select for each family of instruction in the ISA, a representative member to be used instead. The comparison between the throughput of sequences of representative instructions with or without a RAW conflicts will determine the presence or the absence of architectural optimizations.

A second metric to be considered while deriving a microarchitectural model is the execution stage latency, this latency represents the difference in clock cycles between the completion of two consecutive instructions in a sequence of identical instructions. This quantity, for each representative instruction sequence, can be used to derive the latency of each Functional Unit (FU) as well as the number of FUs and whether they are implemented in a pipelined fashion. That is because, in case of a multi-cycle execution, the latency profile of a replicated functional unit is distinguishable from the latency profile of a pipelined functional unit, as clearly visible from Figure 3.5. In the work we present, the metrics just described are processed to build a complete microarchitectural


1 2 3 4 5 6 7 8 9 10 11 12
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB

(a) *Two combinatorial multi-cycle multipliers*


1 2 3 4 5 6 7 8 9 10 11 12
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB
mul r0, r1, r2 IF ID IS EX0 EX1 WB

(b) *Single pipelined multiplier*

Figure 3.5: *Comparison between the execution latency profile of two replicated functional units and a pipelined functional unit while processing a sequence of identical multi-cycle instructions.*

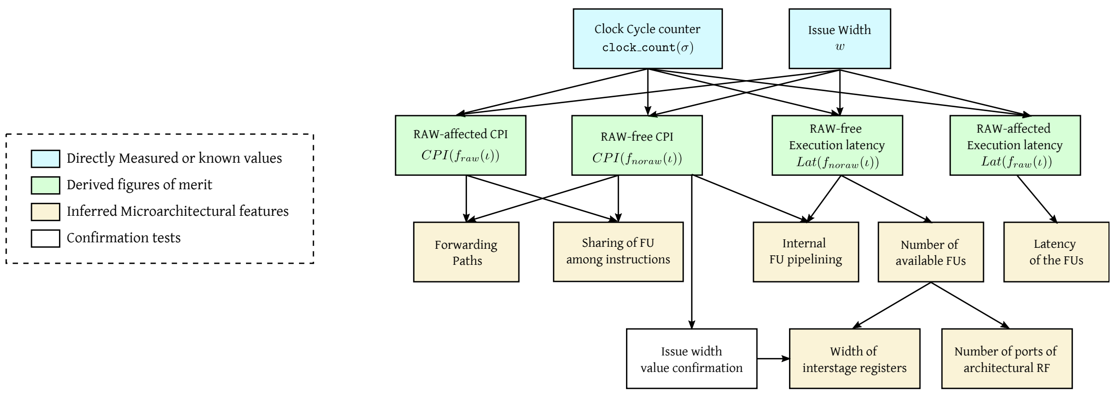

Directly Measured or known values
Derived figures of merit
Inferred Microarchitectural features
Confirmation tests
Clock Cycle counter $\texttt{clock\_count}(\sigma)$
Issue Width w
RAW-affected CPI $CPI(f_{raw}(\iota))$
RAW-free CPI $CPI(f_{noraw}(\iota))$
RAW-free Execution latency $Lat(f_{noraw}(\iota))$
RAW-affected Execution latency $Lat(f_{raw}(\iota))$
Forwarding Paths
Sharing of FU among instructions
Internal FU pipelining
Number of available FUs
Latency of the FUs
Issue width value confirmation
Width of interstage registers
Number of ports of architectural RF

Figure 3.6: *Graph showing the dependencies between the inference steps that were used to obtain a microarchitectural model of the target CPU. Directed arcs from a feature to another indicate that the feature origin of the arc is used to compute the feature destination of the arc.*

model of the execution, able to identify which registers are recombined during the execution of a piece of code. A dependency graph between the features is depicted in Figure 3.6. The CPI and execution latency, computed in both cases of absence and presence of Read After Write conflicts, will allow to derive a complete microarchitectural model, which will comprise the presence of forwarding paths, the sharing of FUs among instructions which is one of the cause of Structural Hazards, the number, latency and pipelining of the employed Functional Units. The number of available functional units will allow to compute the width of the inter-stage register as well as the number of ports of the Register File (RF). In fact, since a higher port count comes with a cost in silicon due to the need of routing multiple data lines, we assume that the maximum number of ports is carefully chosen to fully exploit the processor throughput, but no ports more that the minimum needed for that purpose are placed. We validate each of the identified microarchitectural datapath registers by designing for each register a microbenchmark which causes the overwrite of the register content with a sequence of known values. Therefore, if the register actually exists, it should generate an EM leakage which is correlated with the Hamming Distance of the values that are therein written. We then prepare an EM side channel analysis setup to detect the presence of the expected leakage.

The described technique was applied on two General Purpose COTS CPU: the 4-stage single issue Cortex-M4 [93] and the 5-stage dual issue Cortex-M7 [94]. For the Cortex-M4 we were able to identify 6 datapath synchronous registers, 4 of those were validated with the described side channel leakage. For the Cortex-M7 we identified 12 datapath registers, and 9 of them were successfully validated through the identification of their EM leakage. Furthermore on both CPUs, a CPA attack on an unprotected AES was performed, leading to a complete key recovery with less than 10000 traces on the Cortex-M4 and less than 75000 traces on the Cortex-M7.

3.4.1 Validation Against a Masked AES Implementation

The framework for extracting microarchitectural-level descriptions that was presented in our second publication [23] plays a key role in the development and evaluation of software masked cryptographic implementations. In fact, an effect of the information leakage caused by the micro-architectural registers identifiable with the proposed technique is the leakage of the combination, typically in the form of Hamming Distance (HD), between intermediate values that are not explicitly combined by the semantic of the executed instructions. Examples of such leakages are the HD between two successive values assigned to a register in case of register reuse, or the HD between the values loaded in the Memory Data Register (MDR) during two consecutive memory load operations, even if the two loads are separated by the execution of many non-load instructions. In case of a first order masking implementation, such micro-architectural buffers could lead to the leakage of the correlation between two shares, henceforth invalidating any formal proof of security which is only based on the ISA level. We demonstrate this hypothesis by implementing a first order masked implementation [134]. that never recombines its shares at ISA-level on the Cortex-M4 which exhibited several microarchitectural buffer leakages. Furthermore we show how such an implementation is vulnerable to a first order Correlation Power Attacks (CPA), albeit performed with a larger number of traces than the unprotected implementation.

We implemented the first order masked AES using two sets of masks, the first set $(M^{16}_{IN}, M^{16}_{OUT})$ is composed of one input and one output masks, each one composed of 16 Bytes randomly chosen before each encryption or decryption operation. These 16-Bytes masks are used as input and output masks to protect the AES SubBytes operation by masking the first round Sub-Key with M^{16}_{IN} and replacing the SBox $S(x)$ with $S^{*}(x) = S(x \oplus M^{16}_{IN}) \oplus M^{16}_{OUT}$. Therefore the input of the SubBytes operation will be masked with M^{16}_{IN} and the output will be masked with M^{16}_{OUT} always processing the value in masked form. A second set of masks, (N_{IN}, N_{OUT}), is composed of two single-Byte masks which are applied on all the 16-Bytes of internal AES state and applied as input and output masks for all the operations that reorder the state bytes, comprising the ShiftRows and MixColumns operations. The implementation is executed on a Riscure Pinata evaluation board, that features a STM32F417IGT6 MCU, powered by a Cortex-M4 CPU. The power consumption traces are captured with a Riscure Power Consumption probe, connected to a Tektronix MSO58B oscilloscope. The resulting implementation source code, the captured traces, correlation plots and attack results are publicly hosted on the Zenodo platform [78].

To evaluate the leakage of the target masked implementation we perform a known-key correlation analysis. The knowledge of the key and plaintext allows us to compute the power model values for the intermediate bytes of the first round. We can thus compute the Pearson's Correlation Coefficient between the trace samples and the power model values, which is visible in Fig. 3.7.

A first order attack conducted on the aforementioned traceset [78] allows to retrieve the employed secret key, albeit the presence of a state-of-the-art first order masking scheme.

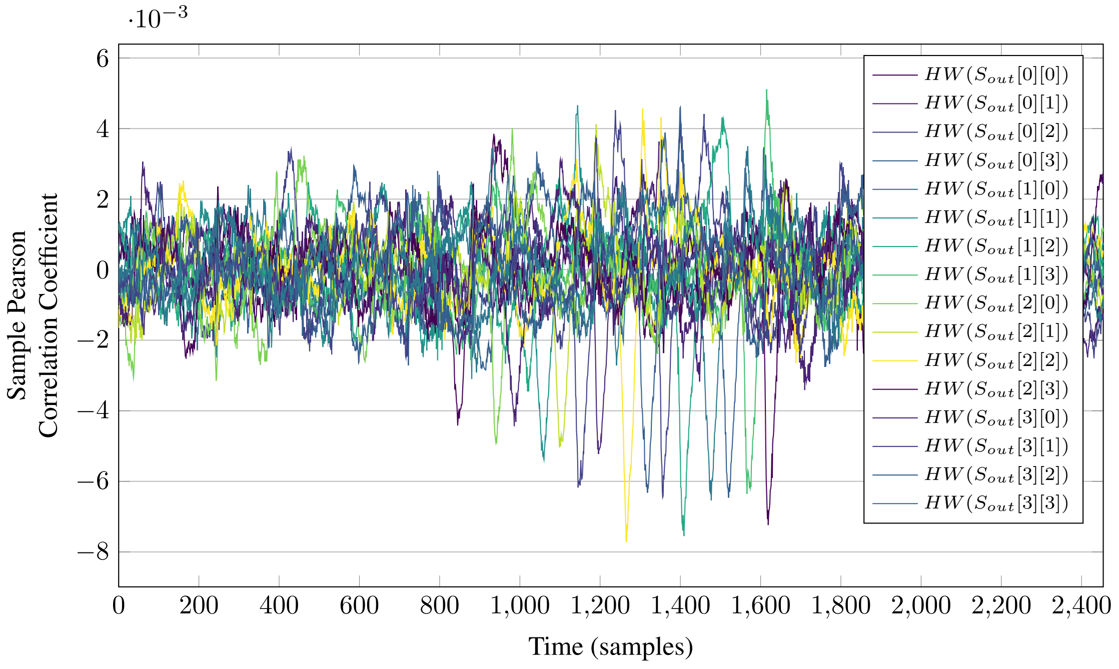


Figure 3.7: *In this figure the Pearson's Correlation Coefficient between the Hamming Weight of each SBox output of the first round and the samples of each trace in the traceset, is plotted for each sample index in each trace. The distinct correlation peaks show that a masking scheme that does not take into account the details of the microarchitectural datapath, produces leakage due to the recombination of its masks in the different datapath registers.*

CHAPTER 4

Conclusions

In the works embedded in this thesis we explored and evaluated four security solutions which solve four different instances of the problem of guaranteeing the confidentiality, integrity and replay protection of data: during the transfer through a link, in a multi-tenant scenario and indirectly by guaranteeing the confidentiality of the key used to encrypt data, when data is holding the security state of a device and against side channel attacks. As a solution to protocol-level physical attacks, a link-level encryption scheme was introduced, able to effectively protect an LPDDR4X bus, guaranteeing confidentiality, integrity and replay protection on the data, command and addresses. The solution has a lightweight resource impact, incurring in a 1.8% average bandwidth penalty and 3.3% power consumption increase, with respect to the unencrypted baseline. The adoption of a link encryption scheme, in opposition to E2E memory encryption, allows to accommodate the requirements of emerging memory, which will leverage differential write strategies to increase the medium endurance. By storing plaintext data, we are able to fully exploit the differential write strategies which are only effective on low-entropy data. Therefore, our solution is able to achieve a zero overhead on the energy and endurance perspective, when considered with respect to emerging memory media.

If the threat model is shifted to include also internal threats, such as malicious code in execution inside other Trusted Virtual Machines or inside the Hypervisor, a solution is required to properly isolate between all the entities. Our solution modifies building blocks commonly found in datacenter-grade servers to implement CXL data confidentiality, integrity and replay-protection. The performance cost is reduced by employing fast Authenticated Encryption with Associated Data (AEAD) ciphers, able to encrypt and generate an integrity tag in a single pass. This approach successfully mitigates against cold-boot style attacks which become way easier with the introduction of emerging memory technologies, that bear data persistence characteristics even at room temperature.

If we consider the mobile ecosystem, the storage of the security state of a device is a well known problem, which received many vulnerable solutions in commercial products, which we describe and expose. We propose an architecture to securely manage the state of a mobile device, guaranteeing to the OEM fine-grained control over the authorization of such security state variations. Our solution leverages commonly available hardware primitives like One-Time-Programmable memory and monotonic counters to enforce five security properties: a cryptographically validated unlock procedure, the management of several intermediate security states, the easy and secure refurbishment of devices, the absence of trust in the storage medium and finally the guarantee that every security state change implies a full device wipe. We propose a security review of the proposed architecture as well as a preliminary investigation on a formal proof of correctness using the Tamarin verifier.

Commercially available CPUs are typically not equipped with hardware side channel countermeasures. On such devices, where software countermeasures are the only available option, the microarchitectural characteristics of the CPU datapath can alter and invalidate the formal security proofs of software side channel masking countermeasures which are unaware of such characteristics. We thus propose a framework for extracting a microarchitectural model of any in-order CPU, which can be used to characterize the datapath of such CPUs. Such details can be exploited to securely implement a masked software cryptographic implementation. Our framework only requires three elements: knowledge about the ISA of the target CPU, a means of measuring the distance between the execution of two instructions in clock cycles, and a side channel setup to measure the information leakage of the hypothesized datapath synchronous registers. Our technique is therefore of general validity and can be applied on commercial CPUs, enabling the implementation of secure masking schemes. In fact we validated the proposed framework on two different commercial CPUs: the Cortex-M4 and the Cortex-M7, obtaining sub-ISA leakage models for both CPUs, that can be used to derive secure software-only side channel masking schemes, that would only be obtainable through a long and error-prone manual validation process to be performed individually for each CPU.

The use of the four proposed security solutions we explored can pave the way to the introduction of emerging memories (like 3D XPoint [106], ReRAM [64], STT-RAM [84], Memristors [124] or ULTRARAM [91]) into data center memory hierarchy and, at the same time, guarantee strong security properties on the stored data. This will enable the adoption of these emerging technologies with the associated lower costs, higher scalability, and persistence, to empower ambitious future Deep Learning and Machine Learning tasks, without sacrificing the security of the processed data.

Bibliography

[1] Advanced Micro Devices. AMD-V™ Nested Paging. Technical report, Advanced Micro Devices, July 2008.

[2] Advanced Micro Devices. AMD Memory Encryption. Technical report, Advanced Micro Devices, April 2016.

[3] Advanced Micro Devices. AMD Memory Guard. Technical report, Advanced Micro Devices, 2020.

[4] Advanced Micro Devices. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. Technical report, Advanced Micro Devices, January 2020.

[5] Advanced Micro Devices. AMD Secure Encrypted Virtualization (SEV). `https://developer.amd.com/sev/`, 2022. (last accessed 2022-01-04).

[6] Advanced Micro Devices, Inc. Nothing Stacks up to EPYC™. `https://www.amd.com/en/products/epyc-server`, 2021. (last accessed 2022-01-02).

[7] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 419–434. USENIX Association, 2020.

[8] Giovanni Agosta, Alessandro Barenghi, and Gerardo Pelosi. A code morphing methodology to automate power analysis countermeasures. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, pages 77–82. ACM, 2012. doi: 10.1145/2228360.2228376.

[9] Giovanni Agosta, Alessandro Barenghi, and Gerardo Pelosi. Compiler-Based Techniques to Secure Cryptographic Embedded Software Against Side-Channel Attacks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(8):1550–1554, 2020. doi: 10.1109/TCAD.2019.2912924.

[10] Giovanni Agosta, Alessandro Barenghi, Gerardo Pelosi, and Michele Scandale. The MEET Approach: Securing Cryptographic Embedded Software Against Side Channel Attacks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 34(8):1320–1333, 2015. doi: 10.1109/TCAD.2015.2430320.

[11] Giovanni Agosta, Alessandro Barenghi, Gerardo Pelosi, and Michele Scandale. Reactive side-channel countermeasures: Applicability and quantitative security evaluation. *Microprocess. Microsystems*, 62:50–60, 2018. doi: 10.1016/j.micpro.2018.07.001.

[12] Alex Cranz. There are over 3 billion active Android devices. `https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021`, 2021. (last accessed 2022-05-16).

[13] Nadav Amit, Muli Ben-Yehuda, Dan Tsafrir, and Assaf Schuster. viommu: Efficient IOMMU emulation. In Jason Nieh and Carl A. Waldspurger, editors, *2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15-17, 2011*. USENIX Association, 2011.

[14] Andreea Florescu. rust-vmm: A Security Journey. `https://kvmforum2021.sched.com/speaker/andreea.florescu15`, September 2021. (last accessed 2022-01-03).

[15] Antmicro. LPDDR4 TEST PLATFORM. `https://antmicro.com/platforms/lpddr4-test-platform/`, 2021. (last accessed 2021-12-27).

[16] Francesco Antognazza, Alessandro Barenghi, and Gerardo Pelosi. Metis: An Integrated Morphing Engine CPU to Protect Against Side Channel Attacks. *IEEE Access*, 9:69210–69225, 2021. doi: 10.1109/ACCESS.2021.3077977.

[17] Ash Turner. How Many Smartphones Are in The World? `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`, 2022. (last accessed 2022-05-14).

[18] N. Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. *IACR Cryptol. ePrint Arch.*, page 163, 2002.

[19] Oran Avraham. Samsung gt-i9300 emmc toolbox. `https://github.com/oranav/i9300_emmc_toolbox`, 2018. (last accessed 2022-05-20).

[20] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic Side-Channel Attacks on Printers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 307–322. USENIX Association, 2010.

[21] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. *IACR Cryptol. ePrint Arch.*, page 413, 2014.

[22] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-only reverse engineering of physical DRAM mappings for rowhammer attacks. In *3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2-4, 2018*, pages 19–24. IEEE, 2018.

[23] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Exploring cortex-m microarchitectural side channel information leakage. *IEEE Access*, 9:156507–156527, 2021.

[24] Alessandro Barenghi, William Fornaciari, Gerardo Pelosi, and Davide Zoni. Scramble Suit: A Profile Differentiation Countermeasure to Prevent Template Attacks. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(9):1778–1791, 2020. doi: 10.1109/TCAD.2019.2926389.

[25] Alessandro Barenghi and Gerardo Pelosi. Side-channel security of superscalar cpus: evaluating the impact of micro-architectural features. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 120:1–120:6. ACM, 2018.

[26] Sean Beaupre. Samdunk: emmc backdoor leading to bootloader unlock on samsung galaxy devices. `http://theroot.ninja/disclosures/SAMDUNK_1.0-03262016.pdf`, March 2016. (last accessed 2022-05-20).

[27] Keith Beckham, David Challener, Arunava Gupta, Joseph Harvilchuck, James Leas, James Lloyd, David Long, Horatio Quinones, Krishna Seshan, and Morris Shatzkes. United states patent: Method and apparatus for causing an open circuit in a conductive line. `https://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=4,962,294.PN.&OS=PN/4,962,294&RS=PN/4,962,294`, 1998. (last accessed 2022-05-23).

[28] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, pages 41–46. USENIX, 2005.

[29] Maxime Rossi Bellom, Damiano Melotti, , and Philippe Teuwen. 2021: A titan m odissey. *Blackhat 2021*, 2021.

[30] Walter Binder. Design and implementation of the J-SEAL2 mobile agent kernel. In *2001 Symposium on Applications and the Internet (SAINT 2001), 8-12 January 2001, San Diego, CA, USA, Proceedings*, pages 35–42. IEEE Computer Society, 2001.

[31] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[32] Robert Buhren, Hans Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. One glitch to rule them all: Fault injection attacks against amd's secure encrypted virtualization. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2875–2889. ACM, 2021.

[33] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. Insecure until proven updated: Analyzing AMD sev's remote attestation. *CoRR*, abs/1908.11680, 2019.

[34] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In David Lie, Mohammad Mannan,

Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 163–177. ACM, 2018.

[35] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[36] Central Intelligence Agency. About The Berlin Tunnel. `https://www.cia.gov/legacy/museum/exhibit/the-berlin-tunnel/`, 2021. (last accessed 2022-01-14).

[37] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. doi: 10.1007/3-540-48405-1_26.

[38] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[39] Yue Chen, Yulong Zhang, Zhi Wang, and Tao Wei. Downgrade attack on trustzone. *CoRR*, abs/1707.05082, 2017.

[40] Zhimin Chen and Yujie Zhou. Dual-rail random switching logic: A countermeasure to reduce side channel leakage. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2006.

[41] Christian Werling, Robert Buhren. Dissecing The AMD Platform Security Processor. `https://media.ccc.de/v/thms-38-dissecting-the-amd-platform-security-processor`, 2019. (last accessed 2022-01-04).

[42] Common Criteria Recognition Arrangement. Common Criteria for Information Technology Security Evaluation. `https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf`, 2017. (last accessed 2021-12-26).

[43] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[44] Intel Corporation. Revolutionizing memory and storage, 2021.

[45] Intel Corporation. Secondary/data drive acceleration with intel® optane™ memory, 2021.

[46] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016.

[47] cxsecurity.com. Xbox 360 Hypervisor Privilege Escalation Vulnerability. Technical report, cxsecurity.com, February 2007.

[48] D. M'Raihi and S. Machani and M. Pei and J. Rydell. TOTP: Time-Based One-Time Password Algorithm. `https://datatracker.ietf.org/doc/html/rfc6238`, 2011. (last accessed 2022-05-16).

[49] Daniel Julius Bernstein. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `https://competitions.cr.yp.to/caesar.html`, 2019. (last accessed 2021-01-07).

[50] Debayan Das, Mayukh Nath, Baibhab Chatterjee, Santosh Ghosh, and Shreyas Sen. STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*, pages 11–20. IEEE, 2019.

[51] Dave Emery. How old is TEMPEST. `https://cryptome.org/tempest-old.htm`, January 2000. (last accessed 2022-01-11).

[52] Wouter de Groot, Kostas Papagiannopoulos, Antonio de la Piedra, Erik Schneider, and Lejla Batina. Bitsliced masking and ARM: friends or foes? *IACR Cryptol. ePrint Arch.*, page 946, 2016.

[53] R.A. DeMillo, D.P. Dobkin, Georgia Institute of Technology, A.K. Jones, and R.J. Lipton. *Foundations of Secure Computation*. Academic Press Rapid Manuscript Reproduction. Academic Press, 1978.

[54] DMTF. Security Protocol and Data Model (SPDM) Specification. Technical report, DMTF, June 2020.

[55] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Technical report, Internet Engineering Task Force, August 2018.

[56] Wikipedia Editors. Comparison of tls implementations, 2021.

[57] Enarx. Enarx. `https://enarx.dev/`, 2021. (last accessed 2022-01-03).

[58] Julie Ferrigno and Martin Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[59] Barry Fitzgerald, Conor Ryan, and Joe Sullivan. An early-life NAND flash endurance prediction system. *IEEE Access*, 9:148635–148649, 2021.

[60] Jacques J. A. Fournier, Simon W. Moore, Huiyun Li, Robert D. Mullins, and George S. Taylor. Security evaluation of asynchronous circuits. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2003.

[61] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[62] Barbara Gigerl, Robert Primas, and Stefan Mangard. Secure and Efficient Software Masking on Superscalar Pipelined Processors. Cryptology ePrint Archive, Report 2021/1110, 2021. .

[63] Barbara Gigerl, Robert Primas, and Stefan Mangard. Secure and efficient software masking on superscalar pipelined processors. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2021.

[64] Olivier Ginez, Jean-Michel Portal, and Christophe Muller. Design and test challenges in resistive switching RAM (reram): An electrical model for defect injections. In *14th IEEE European Test Symposium, ETS 2009, Sevilla, Spain, May 25-29, 2009*, pages 61–66. IEEE Computer Society, 2009.

[65] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture. *CoRR*, abs/2105.03814, 2021.

[66] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptol. ePrint Arch.*, page 204, 2016.

[67] Sylvain Guilley, Philippe Hoogvorst, Yves Mathieu, Renaud Pacalet, and Jean Provost. CMOS structures suitable for secured hardware. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*, pages 1414–1415. IEEE Computer Society, 2004.

[68] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008.

[69] Michael Henson and Stephen Taylor. Beyond full disk encryption: Protection on security-enhanced commodity processors. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2013.

[70] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is not good enough - deriving optimal distinguishers from communication theory. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2014.

[71] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3). Technical report, Division of Mathematical Sciences, Nanyang Technological University, September 2016.

[72] A. Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. Xenatera Press, 2003.

[73] Igor Skochinsky, Nicola Corna. Intel ME: Myths and reality. `https://media.ccc.de/v/34c3-8782-intel_me_myths_and_reality`, 2017. (last accessed 2022-01-03).

[74] Intel Corporation. Intel® Trust Domain Extensions. Technical report, Intel Corporation, September 2020.

[75] Intel Corporation. Intel® Architecture Memory Encryption Technologies. Technical report, Intel Corporation, April 2021.

[76] Intel Corporation. Intel® Virtualization Technology (Intel® VT). Technical report, Intel Corporation, 2021.

[77] Niccolò Izzo, Alessandro Barenghi, Luca Breveglieri, Gerardo Pelosi, and Paolo Amato. A secure and authenticated host-to-memory communication interface. In Francesca Palumbo, Michela Becchi, Martin Schulz, and Kento Sato, editors, *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, pages 386–391. ACM, 2019.

[78] Niccolò Izzo. Pinata masked aes round 1, September 2022.

[79] JC-42.3B Committee. DDR5 SDRAM. Technical report, JEDEC SOLID STATE TECHNOLOGY ASSOCIATION, October 2021.

[80] JC-42.6 Committee. LPDDR4 SDRAM. Technical report, JEDEC SOLID STATE TECHNOLOGY ASSOCIATION, February 2021.

[81] JC-64.1 Committee. Embedded MultiMediaCard (e•MMC) e•MMC/Card Product Standard. Technical report, JEDEC SOLID STATE TECHNOLOGY ASSOCIATION, March 2010.

[82] JC-64.1 Committee. Universal Flash Storage (UFS). Technical report, JEDEC SOLID STATE TECHNOLOGY ASSOCIATION, January 2018.

[83] Paul A. Karger and A. J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 29 - May 2, 1984*, pages 2–12. IEEE Computer Society, 1984.

[84] Takayuki Kawahara, Kenchi Ito, Riichiro Takemura, and Hideo Ohno. Spin-transfer torque RAM technology: Review and prospect. *Microelectron. Reliab.*, 52(4):613–627, 2012.

[85] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Rowhammer: Reliability analysis and security implications. *CoRR*, abs/1603.00747, 2016.

[86] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[87] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[88] Markus G. Kuhn and Ross J. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In David Aucsmith, editor, *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*, volume 1525 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 1998.

[89] Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. A comprehensive performance analysis of hardware implementations of CAESAR candidates. *IACR Cryptol. ePrint Arch.*, page 1261, 2017.

[90] Jin Baek Kwon. On bypassing page cache for block devices on storage class memory. In James Jong Hyuk Park, Shu-Ching Chen, and Kim-Kwang Raymond Choo, editors, *Advanced Multimedia and Ubiquitous Engineering - MUE/FutureTech 2017, Seoul, Korea, 22-24 May 2017*, volume 448 of *Lecture Notes in Electrical Engineering*, pages 361–366, 2017.

[91] D. Lane, P. D. Hodgson, R. J. Potter, R. Beanland, and M. Hayne. Ultraram: Toward the development of a iii–v semiconductor, nonvolatile, random access memory. *IEEE Transactions on Electron Devices*, 68(5):2271–2274, 2021.

[92] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In Larry Rudolph and Anoop Gupta, editors, *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000*, pages 168–177. ACM Press, 2000.

[93] ARM Limited. Cortex-M4 Instruction Set Summary. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf`, 2010.

[94] ARM Limited. ARM® Cortex®-M7 Processor Technical Reference Manual. https://documentation-service.arm.com/static/5e906b038259fe2368e2a7bb?token=, 2014.

[95] Victor Lomné, Philippe Maurine, Lionel Torres, Michel Robert, Rafael Soares, and Ney Calazans. Evaluation on FPGA of triple rail logic robustness against DPA and DEMA. In Luca Benini, Giovanni De Micheli, Bashir M. Al-Hashimi, and Wolfgang Müller, editors, *Design, Automation and Test in Europe, DATE 2009, Nice, France, April 20-24, 2009*, pages 634–639. IEEE, 2009.

[96] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography. *ACM Comput. Surv.*, 54(6):122:1–122:37, 2021.

[97] lowRISC contributors. Opentitan is the first open source project building a transparent, high-quality reference design and integration guidelines for silicon root of trust (rot) chips. https://opentitan.org/, 2022. (last accessed 2022-05-23).

[98] F. Mace, F. x. St, I. Hassoune, J. d. Legat, and J. j. Quisquater. A dynamic current mode logic to counteract power analysis attacks. In *In The Proceedings of DCIS 2004*, pages 186–191, 2004.

[99] Nicholas Mainardi, Alessandro Barenghi, and Gerardo Pelosi. Plaintext recovery attacks against linearly decryptable fully homomorphic encryption schemes. *Comput. Secur.*, 87, 2019.

[100] Moshe Malka, Nadav Amit, Muli Ben-Yehuda, and Dan Tsafrir. riommu: Efficient IOMMU for I/O devices that employ ring buffers. In Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas, editors, *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2015, Istanbul, Turkey, March 14-18, 2015*, pages 355–368. ACM, 2015.

[101] Maxim Goyachy and Mark Ermolov. How to hack a turned-off computer or running unsigned code in intel Management Engine. https://www.blackhat.com/eu-17/briefings.html#how-to-hack-a-turned-off-computer-or-running-unsigned-code-in-intel-management-engine, 2017. (last accessed 2022-01-14).

[102] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). *IACR Cryptol. ePrint Arch.*, page 193, 2004.

[103] Thomas S. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois at Chicago, USA, 01 2000.

[104] Lauren De Meyer, Elke De Mulder, and Michael Tunstall. On the effect of the (micro)architecture on the development of side-channel resistant software. *IACR Cryptol. ePrint Arch.*, page 1297, 2020.

[105] Michael Steil, Felix Domke. The Xbox 360 Security System and its Weaknesses. https://www.youtube.com/watch?v=uxjpmc8ZIxM, 2008. (last accessed 2021-12-29).

[106] Micron. Breakthrough nonvolatile memory technology, 2018.

[107] Tilo Müller, Felix C. Freiling, and Andreas Dewald. TRESOR runs encryption securely outside RAM. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[108] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. Enabling practical processing in and near memory for data-intensive computing. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 21. ACM, 2019.

[109] Ruth Nelson and John H. Heimann. SDNS architecture and end-to-end encryption. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 356–366. Springer, 1989.

[110] NVM Express, Inc. NVM Express® Base Specification. Technical report, NVM Express, Inc., July 2021.

[111] Institute of Electrical and Electronics Engineers. Ieee standard for local and metropolitan area networks-media access control (mac) security. *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pages 1–239, 2018.

[112] Oran Avraham. eMMC hacking, or: how I fixed long-dead Galaxy S3 phones. https://media.ccc.de/v/34c3-8784-emmc_hacking_or_how_i_fixed_long-dead_galaxy_s3_phones, 2017. (last accessed 2022-05-20).

[113] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. *IACR Cryptol. ePrint Arch.*, page 345, 2017.

[114] DongGook Park, Colin Boyd, and Sang-Jae Moon. Forward secrecy and its application to future mobile communications security. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings*, volume 1751 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2000.

[115] PCI-SIG. Single Root I/O Virtualization and Sharing Specification Revision 1.1. Technical report, PCI-SIG, January 2010.

[116] PCI-SIG. PCI Express Base Specification Revision 5.0, Version 1.0. Technical report, PCI-SIG, May 2019.

[117] PCI-SIG. Integrity and Data Encryption (IDE) ECN. Technical report, PCI-SIG, November 2020.

[118] pmem.io. Persistent memory is revolutionary, 2021.

[119] Positive Technologies. Positive Technologies: Unfixable vulnerability in Intel chipsets threatens users and content rightsholders. `https://www.ptsecurity.com/ww-en/about/news/unfixable-vulnerability-in-intel-chipsets-threatens-users-and-content-rightsholders/`, 2020. (last accessed 2022-01-14).

[120] Positive Technologies. Positive Technologies Discovers Vulnerability in Intel Processors used in Laptops, Cars and Other devices. `https://www.ptsecurity.com/ww-en/about/news/positive-technologies-discovers-vulnerability-in-intel-processors-used-in-laptops-cars-` 2021. (last accessed 2022-01-14).

[121] postmarketOS.org contributors. A real Linux distribution for phones and other mobile devices. `https://postmarketos.org/`, 2022. (last accessed 2022-05-09).

[122] The LineageOS Project. Lineageos statistics. `https://www.lineageoslog.com/statistics`, 2022. (last accessed 2022-05-23).

[123] Rui Qiao and Mark Seaborn. A new approach for rowhammer attacks. In William H. Robinson, Swarup Bhunia, and Ryan Kastner, editors, *2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016*, pages 161–166. IEEE Computer Society, 2016.

[124] V. Ravi, Shivendra Singh, and S. Sofana Reka. Memristor-based 2d1m architecture: Solution to sneak paths in multilevel memory. *Trans. Emerg. Telecommun. Technol.*, 32(1), 2021.

[125] Alin Razafindraibe, Michel Robert, and Philippe Maurine. Analysis and improvement of dual rail logic as a countermeasure against DPA. In Nadine Azémard and Lars J. Svensson, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, 17th International Workshop, PATMOS 2007, Gothenburg, Sweden, September 3-5, 2007, Proceedings*, volume 4644 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2007.

[126] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 1–18. USENIX Association, 2016.

[127] Red Hat, Inc. Spice Protocol. `https://www.spice-space.org/spice-protocol.html`, 2009. (last accessed 2022-01-03).

[128] A. Redaelli. *Phase Change Memory: Device Physics, Reliability and Applications*. Springer International Publishing, 2017.

[129] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015. doi: 10.1007/978-3-662-47989-6_37.

[130] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010. doi: 10.1007/978-3-642-15031-9_28.

[131] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. A side journey to titan. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 231–248. USENIX Association, 2021.

[132] Brian Rogers, Yan Solihin, and Milos Prvulovic. Memory predecryption: hiding the latency overhead of memory encryption. *SIGARCH Comput. Archit. News*, 33(1):27–33, 2005.

[133] Rusty Russell. virtio: towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Oper. Syst. Rev.*, 42(5):95–103, 2008.

[134] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.

[135] Hermann Seuschek, Fabrizio De Santis, and Oscar M. Guillen. Side-channel leakage aware instruction scheduling. In *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC 2017, Stockholm, Sweden, January 24, 2017*, pages 7–12, 2017.

[136] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.

[137] Dr. Debendra Das Sharma and Siamak Tavallaei. Compute Express Link™ 2.0 White Paper. Technical report, CXL Consortium, November 2021.

[138] Patrick Simmons. Security through amnesia: A software-based solution to the cold boot attack on disk encryption. *CoRR*, abs/1104.4843, 2011.

[139] Danil Sokolov, Julian P. Murphy, Alexandre V. Bystrov, and Alexandre Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Trans. Computers*, 54(4):449–460, 2005.

[140] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. Universal 2nd Factor (U2F) Overview. Technical report, FIDO Alliance, April 2017.

[141] Michael Steil. 17 mistakes Microsoft made in the Xbox security system. In Chaos Communication Congress, editor, *Proceedings of the 22nd Chaos Communication Congress*, 2005.

[142] Lifeng Su, Stephan Courcambeck, Pierre Guillemin, Christian Schwarz, and Renaud Pacalet. Secbus: Operating system controlled hierarchical page-based memory bus protection. In Luca Benini, Giovanni De Micheli, Bashir M. Al-Hashimi, and Wolfgang Müller, editors, *Design, Automation and Test in Europe, DATE 2009, Nice, France, April 20-24, 2009*, pages 570–573. IEEE, 2009.

[143] Synopsys, Inc. The heartbleed bug. `https://heartbleed.com/`, 2021. (last accessed 2021-12-24).

[144] The GnuPG Project. The GNU Privacy Guard. `https://gnupg.org/`, 2021. (last accessed 2022-01-02).

[145] The gVisor Authors. gVisor is an application kernel for containers that provides efficient defense-in-depth anywhere. `https://gvisor.dev/`, 2021. (last accessed 2022-01-03).

[146] The LineageOS Project. LineageOS Android Distribution. `https://lineageos.org/`, 2022. (last accessed 2022-05-09).

[147] The OpenSSL Project Authors. OpenSSL, Cryptography and SSL/TLS Toolkit. `https://www.openssl.org/`, 2021. (last accessed 2021-12-26).

[148] The QEMU Project Developers. QEMU Security. `https://qemu-project.gitlab.io/qemu/system/security.html`, 2021. (last accessed 2022-01-08).

[149] Tim Dierks, Christopher Allen. The TLS Protocol Version 1.0. Technical report, Internet Engineering Task Force, March 1997.

[150] Wim van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Comput. Secur.*, 4(4):269–286, 1985.

[151] Neal H. Walfield and Werner Koch. TOFU for openpgp. In Michalis Polychronakis and Cristiano Giuffrida, editors, *Proceedings of the 9th European Workshop on System Security, EUROSEC 2016, London, UK, April 18-21, 2016*, pages 2:1–2:6. ACM, 2016.

[152] Matthew Wilcox. Dax: Page cache bypass for filesystems on memory storage, 2014.

[153] P. Wright and P. Greengrass. *Spycatcher*. Mandarin, 1989.

[154] Jun Yang, Youtao Zhang, and Lan Gao. Fast secure processor for inhibiting software piracy and tampering. In *Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, CA, USA, December 3-5, 2003*, pages 351–360. IEEE Computer Society, 2003.

[155] Davide Zoni, Alessandro Barenghi, Gerardo Pelosi, and William Fornaciari. A Comprehensive Side-Channel Information Leakage Analysis of an In-Order RISC CPU Microarchitecture. *ACM Trans. Design Autom. Electr. Syst.*, 23(5):57:1–57:30, 2018. doi: 10.1145/3212719.

CHAPTER 5

Published Works

A Secure and Authenticated Host-to-Memory Communication Interface

Niccolò Izzo
Alessandro Barenghi
niccolo.izzo@polimi.it
alessandro.barenghi@polimi.it
Politecnico di Milano

Luca Breveglieri
Gerardo Pelosi
luca.breveglieri@polimi.it
gerardo.pelosi@polimi.it
Politecnico di Milano

Paolo Amato
Micron Technology Inc.
pamato@micron.com

ABSTRACT

Emerging non-volatile memories (NVMs) have the potential to change the memory-storage hierarchy in computing devices, and even to replace DRAM as main memories. In fact NVMs, beside offering byte-addressability and data persistence, promise better scalability and higher capacity than DRAM. However, from a security point of view, the persistent nature of emerging memories provides a larger time window to exfiltrate data from a device with respect to current DRAM-based main memories, and NVMs have in general lower write endurance than DRAM, thus requiring wear-out conscious encryption schemes. In this work we propose an architectural solution to secure non-volatile emerging memories, providing confidentiality, integrity and authenticity to the entire set of data, addresses and commands. Our solution relies on securing and authenticating the entire information transport between the host controller and the memory, enabling the storage of cleartext data inside the NVM. Such an approach allows to retain the advantage of differential write strategies without forsaking security. We validate our proposed architecture through the simulation of a set of software benchmarks on an embedded architecture, employing the gem5 trace-based architectural simulator.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**; **Embedded systems security**; *Hardware attacks and countermeasures*; Distributed systems security.

KEYWORDS

Secure memories, Embedded security, Memory encryption, Emerging memories, Security protocols

ACM Reference Format:
Niccolò Izzo, Alessandro Barenghi, Luca Breveglieri, Gerardo Pelosi, and Paolo Amato. 2019. A Secure and Authenticated Host-to-Memory Communication Interface. In *Proceedings of the 16th conference on Computing Frontiers (CF '19), April 30-May 2, 2019, Alghero, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3310273.3323401

1 INTRODUCTION

Main memory subsystems are critical components of every computing platform, ranging from high performance systems such as servers and desktops, to resource constrained ones such as sensors, embedded and mobile systems. At the same time, the increasing demand for powerful and cost effective devices pushes forward the research and development of memory subsystems able to scale in terms of size and power consumption, beyond the limits of already available technologies. Emerging Non-Volatile Memory (NVM) technologies, such as Phase Change Memory (PCM) [20], 3D XPoint (3DXP) [16], Spin-Transfer Torque RAM (STT-RAM), and Memristors have gained great attention from both academia and industry; among them PCM and 3DXP have already hit the market.

Emerging technology may be employed as non volatile storage with access latencies and write endurance surpassing the performance of the traditionally available options, i.e., Flash memories and electro-mechanical hard disks. The ability of performing byte-addressable accesses to the stored information, as well as a low access latency, makes NVMs also a viable candidate to be paired with traditional DRAM based main memory, or to replace them altogether. In either case, this integration will lead to the ability to process persistent data with figures of merit by far surpassing the current solutions, in terms of scalability and energy consumption.

Together with the increase in computational efficiency, NVMs also provide new challenges in ensuring security and privacy properties on the stored data. As an example, the use of NVMs as a further layer of the memory hierarchy has the potential to decrease the effort required to perform the so-called *cold boot* attacks [8], due to the extended persistence of the stored data with respect to the typically lower persistence of DRAM technology. Such a problem may be exacerbated in an IoT scenario, where it is common practice to store sensitive data in cleartext on the persistent storage.

Finally, the NVM security implementation strategies should be able to match tight performance constraints in terms of access latency and throughput, so to avoid a significant reduction of the advantage of employing NVMs. In particular, such implementation strategies should take into account the constraints imposed by the write endurance of the NVM technologies in their design, in order to preserve their practical usability.

Contributions. In this work, we propose an architectural solution to secure NVMs, made of two modules, one on the host controller and one on the NVM module itself. We aim at providing a nearly drop-in solution inserting both modules in the form of a Dual-Data-Rate (DDR) Physical (PHY) Interface (DDR-PHY or DFi in short) to DFi adapter. The component in the NVM module is designed to be small enough to be area-compatible with the resource figures of

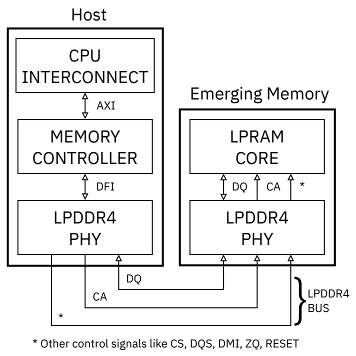


Figure 1: Block diagram of a main memory subsystem based on Emerging Memory, connected with an LPDDR4 Bus. AXI and DFi are two interconnection protocols, DQ, CA and * are signals belonging to the LPDDR4 protocol.

modern DDR memory controllers. The two modules cooperate to ensure confidentiality, integrity and authentication on transferred commands, data and addresses, allowing the NVM module to store cleartext data. We validate the performance of our proposed architecture simulating the execution of the MiBench2 IoT benchmark suite [10] on the gem5 trace-based simulator [3], paired with with the nvmain main memory simulator [18].

2 BACKGROUND

After providing a set of specifications for a plausible emerging memory technology, we describe our considered attacker model.

2.1 Emerging Memory Subsystem

To be able to provide a sufficiently optimized solution, we need to pick one emerging NVM technology to tailor our work to. In particular, we are going to consider PCM technology. In fact, on one hand, PCM is making its way into the embedded devices market [21], on the other hand, this technology has undergone a long refinement process since its market introduction in 2008, so it is mature enough to be considered for our analysis. PCM memories store each bit of information as the phase of a small chalcogenide cell, the phase is switched by heating up the cell, and controlling the cooling time. If the cell is cooled quickly, it remains in its amorphous form; otherwise if cooled slowly it forms a crystal lattice. The two phases have different resistivity, which can be sensed to perform data reads.

This technology is viable to realize a scalable energy-efficient memory, although write operations are slower and more energy demanding than read operations. As a consequence, several encoding techniques like Data Inversion (DI), which stores the word in plain on inverted form to reduce the number of ones, and Data Comparison Write (DCW), which rewrites only the bits whose values have changed when performing in-place writes, are employed. These techniques work best with low-entropy data, when storing high-entropy such as encrypted data, the number of written bits is on average the half of the logical word size.

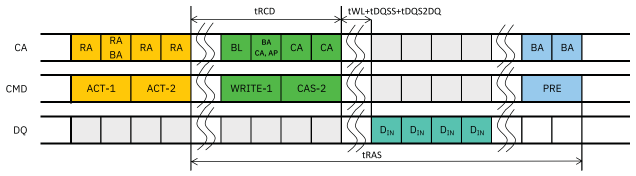


Figure 2: LPDDR4X Write Transaction Timing Chart. RA=Row Address, BA=Bank Address, [ACT-1, ACT-2]=Activate Commands, BL=Burst Length, CA=Column Address, CAS-2=Column Address Strobe, D_{IN}=Data Input, PRE=Precharge

We consider a hypothetical PCM based memory architecture, based on the LPDDR4X communication bus. As is depicted in Figure 1, common internal interconnection protocols were chosen such as Advanced eXtensible Interface (AXI) to interconnect the CPU and Memory Controller, and DFi (logical form of LPDDR4X PHY) to attach the controller to the PHY. These internal protocols have the sole purpose of giving more likelihood to the architecture, our solution is not specific or strictly bound to those protocols.

Memory Interface Low Power Double Data Rate v.4 eXtended (LPDDR4X) is a recent synchronous DRAM specification standard [12]. Its adoption is rapidly growing for mobile, laptops and Internet of Things devices, due to its low power features and its ability to reach a maximum 21.3 GB/s transfer rate by processing up to four data words within a clock cycle; those features make it the most plausible candidate for a PCM-based memory.

Among the signals of the LPDDR4X interconnect, we will consider: the 6-bit wide *Command/Address Inputs* (CA), which provides commands and addresses, the 16-bit *Data Input/Output* (DQ), which is a bidirectional half-duplex data bus, and the single-bit *Chip Select* CS, which is also part of the command code. A full description of the clock signals and power rails, can be found on the LPDDR4X specification document [11, 13]. The LPDDR4X standard is characterized by considerable delays to be enforced between the various commands during the memory operations. As an example, in Figure 2 it is depicted a complete write sequence. The time between the end of the Row Address Strobe (RA orange slots) and the beginning of the data transfer (D_{IN} cyan slots) is at least 27ns (22 clock cycles at 800 MHz). Therefore free headroom is present on the command bus among different operations.

2.2 Scenario and Attacker Model

The pervasive nature of IoT devices exposes them to theft or seizure by an adversary. This fact, combined with the confidentiality of the data stored on them, makes them valuable targets for attackers.

Some IoT devices may be equipped with a JTAG or DMA-based interface, which provides indirect access to the main memory. A simple solution to this threat model is to disable those interfaces for production devices: a realistic scenario for IoT, where external DMA devices are rarely used. Cold-boot may allow an attacker to execute arbitrary code on target platforms and obtain memory access, however this attack scenario has already been solved in commercial devices by employing a hardware-based Chain of Trust [14], which verifies the integrity of each boot component up to the OS kernel.

An attacker may instead attempt to extract the contents of the non-volatile memory either de-soldering it from the main board and directly trying to access it, or inserting an interposer [22] to perform wiretapping and tampering of the host-memory transactions. Finally, the attacker may be willing to swap the persistent emerging memory between the device under attack, and another instance of the same model, which is under the attacker's control. We note that such attacks are expected to become simpler in case the emerging memory is designed to be removable, as it may happen in the future.

Through the aforementioned techniques, an attacker will be able to observe and alter commands, addresses and data of the operations that the host performs on its memory. We will now analyze the impact of such actions on the overall security of the system.

A breach in the commands integrity, even in the assumption that addresses and data are kept confidential and pristine, may still have a significant impact from a security perspective. Whilst open literature does not report evidence of attacks conducted with command alterations yet, changing a write command into a read one has the practical effect of not performing the write action itself. This can be exploited to prevent the storage of refreshed cryptographic keys. As described by Gueron [7], violating command integrity can cause security-affecting alterations, even if the attacker has no visibility or control on the altered data (Blinded Random Block Corruption), as it may corrupt cryptographic keys, or public RSA keys employed to validate signatures, thus allowing forgeries due to the ease of factorization of some malformed moduli [4]. A breach in command authenticity may also allow an attacker to replay non-fresh commands effectively rolling back security updates, with a clear adverse effect on the security of the system.

A violation on address confidentiality alone is enough to perform access pattern attacks, thus obtaining the secret key information in software cipher implementations as demonstrated by Osvik et al. [17]. If the integrity of addresses is also breached, the attacker will be able to bypass the Operating System access control mechanisms, e.g., writing data onto a kernel page table, and consequently to perform privilege escalation. If the address authenticity is violated, it is possible for an attacker to perform splicing attacks among memory transactions. This can be exploited in practice to hijack memory writes, achieving the same effects of data corruption possible via command integrity violations.

A breach in data confidentiality, integrity or authenticity has the most evident impact on both the security of an embedded system, and the assets that should be protected, i.e., the gathered data on the said system.

Coherently with the attacker capabilities described, and the adverse effects on the security of the system, we thus summarize the threats that our proposal prevents as: i) *command authenticity and integrity violations*, ii) *address confidentiality, integrity and authenticity violations*, iii) *transferred data confidentiality, integrity and authenticity violations*.

3 A SECURE LPDDR4X ADAPTER

In this section we state the security properties we provide with our proposal, and describe our architectural solution, which takes the form of an LPDDR4X adapter architecture, which can be integrated on the host SoC DFi interface and on the main memory device silicon. Our aim is to prevent the security property violations described in Section 2. In this scenario we want to guarantee the confidentiality of data and addresses, and the integrity (including the freshness) and authenticity of data, addresses and commands. Due to the high data rates required by the LPDDR4X bus, we chose to provide the authenticity and integrity of data and commands exploiting symmetric primitive based authentication mechanisms. Combining this with the need of providing confidentiality on addresses and data, led us to choose a symmetric cipher operating in an authenticated mode of encryption. One authenticated cipher will be applied to the bidirectional data traffic, and another will be applied to the CA signals carrying commands and addresses. The idle times mandated by the LPDDR4X protocol will be used to transmit the authentication tag resulting from the authenticated encryption. This will forbid any attempt to alter encrypted data.

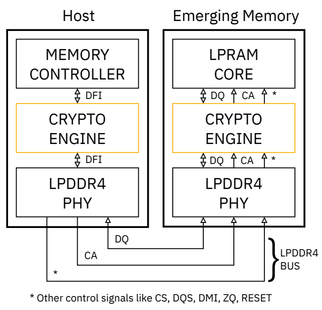


Figure 3: Block scheme depiction of our secure adapter. The cryptographic engines are highlighted in yellow

3.1 Key management

Employing a symmetric cipher with an authenticated mode of encryption requires the communicating endpoints to share a common secret. While permanently storing a symmetric key in both the host and the memory would be sufficient to thwart online attacks led with a bus interposer, such an approach would be ineffective against an attacker which employs a second fully controlled device to perform offline attacks. In fact if the memory manufacturer uses a single key for all the produced devices, once leaked, all the devices using the same key will become vulnerable. Another option in general is to generate keys in the production line and burn them into both the memory and the SoC. This scenario however poses a higher responsibility on the manufacturer and on its employees. In fact the keys could be collected at the production line and sold to malicious agents.

We thus propose to agree a symmetric key between the host and the main memory, during the first boot of the device, in the hands of the final customer. Most modern SoCs are equipped with a True Random Number Generator (TRNG), which should be used to generate the 128 bit key to be transferred to the memory device. Since the key exchange is expected to take place in a trusted environment, the key may be transferred through the LPDDR4X bus as cleartext. Indeed it is sufficient to have the main memory retain the information of being paired to a host, in the form of the received

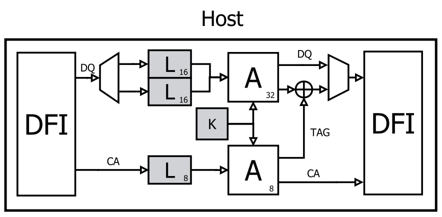


Figure 4: LPDDR4X Adapter Internal Architecture: Memory Controller DFi interface (*left*), and LPDDR4 PHY DFi interface (*right*). "A" blocks are Acorn-8 and Acorn-32 engines. "L" blocks are synchronous latches, "K" is the ephemeral key, and TAG is the Acorn authentication tag

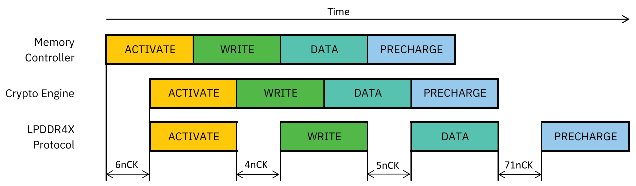


Figure 5: Encrypted Write Transaction Timing Diagram: the encryption latency of 6nCK cannot be hidden, however the protocol is characterized by higher delays between its phases

key, so that the device will accept to perform the pairing only if no keys are found, Once the initial key agreement is completed, the main memory will only communicate employing the agreed key, thus effectively preventing device-swapping attacks.

A key erasure procedure is often required by manufacturers to perform an integrate decommissioning of the memory. However the procedure should be triggered only through a dedicated command authenticated with vendor keys. Such a command should delete the entire memory contents, and subsequently erase the communication key itself. In this fashion, the memory module can either be securely decommissioned, or can be repurposed to fit on a different device without breaches of the security guarantees.

3.2 Proposed architecture

We designed a cryptographic architecture to guarantee integrity and confidentiality on addresses and data, and integrity only on commands, which is depicted in Figure 3. This architecture is derived from the one in Figure 1 by inserting a pair of DFi-to-DFi LPDDR4X compatible cryptographic adapters, just before the PHY interfaces in both the host and the Emerging Memory. Our architecture is focused on guaranteeing the aforementioned properties, without limiting the performance or endurance of emerging memories, and adapting to the energy-aware context of mobile systems.

As it is described in Section 3.1, our host and memory already possess a reasonably large (128-bit) shared cryptographic key. We employ a symmetric ephemeral key to avoid replay attacks upon the ciphers reset. This key is used for the authenticated encryption of data, addresses and commands, as can be seen in Figure 4.

The two ciphers are clocked with the same clock signal that drives the LPDDR4X interface, so that they are synchronous to the two interfaces, thus no Phase-Locked Loop (PLL) circuitry is needed; moreover no buffers are required and all the signals are encrypted in real-time, as depcted in figure 5. Only two 16-bit latch arrays for data are necessary to condense two 16-bit data words in a single 32-bit plaintext block. An 8-bit latch register is applied on the CA bus to keep the DQ and CA signals synchronized. The tags resulting from the authenticated encryption of the DQ and CA signals are XORed together and transferred on the DQ bus during the idle times of the protocol; these tags are filtered out by the cryptographic engine, to avoid protocol anomalies.

Symmetric Cipher Selection Choosing the cipher primitive to employ for authenticated encryption is a sensitive matter, because the embedded environments mandate strict requirements in terms of silicon area, performance and power consumption.

The Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [2] produced some interesting candidates for small and lightweight implementation of authenticated encryption. Among them we selected ACORN [23] for its small area footprint and high throughput characteristics. According to Kumar et al. [15], its 8-bit wide parallel implementation reaches 9.09 Gb/s occupying 9469.8 mm^2, while the 32-bit wide version at 34.04 Gb/s, using TSMC 65 mm technology.

We use the 32-bit implementation to encrypt two 16-bit samples of the DQ signals, and the 8-bit implementation to encrypt the 7-bit CA bus. ACORN has a 293-bit internal state, which is first filled with the secret key and an Initialization Vector (IV), then the cipher must be initialized by running it with zero inputs for 1792 steps. The initialization procedure is time consuming, so we leverage the fact that the ACORN input influences the cipher to avoid incrementing the shared counter at every sent message. In fact even two consecutive identical transactions would result in two different ciphertexts. ACORN however can encrypt a maximum of 2^{64} bits of plaintext, when this threshold is reached, the counter must be incremented and the cipher re-initialized. ACORN can produce a 128-bit authentication tag at the end of each message encryption, sending the tag over the bus requires 4 clock cycles (nCK), leveraging the 16-bit DDR data bus.

Protected architecture bus operation We now assume that the two endpoints already possess a shared secret. When the two devices are powered up, the host derives a 128 bit ephemeral key which is used as a counter and transmitted after encryption to the memory, the infrastructure for the encryption of the counter is trivial and is not depicted in the architecture scheme. This counter is incremented at each memory system initialization and is kept synchronized among the two parties. The counters are exchanged again if the number of transferred plaintext bits reaches 2^{64} and every time the cipher is initialized, to avoid replay attacks.

After the system initialization, the two cryptographic endpoints encrypt every message that is sent through them, and at the end of each data burst, they synchronize the XORed authentication TAGs in cleartext over the DQ bus as shown in the top right quarter of Figure 4. In case of failed verification, no information about the wrong tag or ciphertext has to be leaked to the attacker, to be safe from known-plaintext attacks and chosen-ciphertext attacks. Instead the system is reset to impede potential tampering attempts. To lower our performance cost, we modify the ACORN cipher to emit the plaintext in real time, without waiting for the tag verification. This can be considered safe since the ciphertext is decrypted in the silicon, thus the resulting plaintext cannot be obtained by an attacker. Furthermore, a rollback mechanism must be implemented to impede the storage of erroneous data, for example by holding the `PRECHARGE` commands until a new row is activated (`ACT`) on the same bank. In case verification errors occur, during a read operation, the anomaly can be signalled to the Kernel through the use of an interrupt. The signaling of verification failures can be useful both as a secondary attack countermeasure and as a LPDDR4X reliability figure, to prevent channel-based memory errors.

In the LPDDR4X standard the `DQ` bus is bidirectional half-duplex, therefore the ACORN cipher must be able to interleave encryption and decryption without re-initialization. This is possible since ACORN has a stream cipher structure, therefore its encryption and decryption operations are identical. The whole encryption process only adds 4 clock cycles (nCK) to each memory transaction. An upper bound on the resulting overhead can be statically computed as the 10% of the transaction time; the actual overhead will be smaller for longer burst transactions.

4 EXPERIMENTAL EVALUATION

To carry out an energy and performance evaluation of our system, we need to place our selected technologies into a fully detailed embedded device model. We report an overview of the hypothesized memory subsystem in Figure 1.

The device is based on an integrated System on Chip (SoC), which contains the CPUs, the CPU interconnect fabric, a memory controller connected to the fabric via AXI bus, and a LPDDR4X PHY connected to the latter via DFi interconnection [5]. We consider as plausible specifications for the CPU, an ARMv7-a 32-bit processor, clocked at 1 GHz, with 32 KB L1 instruction cache, 32 KB L1 data cache and 256 KB of shared L2 cache. Finally, the LPDDR4X PHY is connected to a BGA socket to allow Package on Package (PoP) mount of the main memory package. We produced our figures of merit using gem5 [3] as a trace-based syscall-level emulator, and using NVMain [18] as an emerging memory simulator. Relevant metrics are extracted from the execution of the MiBench2 [10] benchmark suite, which is commonly used for IoT devices performance analyses.

Connected to the LPDDR4X socket, there is a PCM-based main memory device, which we refer to as *Low-power Phase-change Random Access Memory* (LPRAM), it consumes 25 pJ/bit for each read and 100 pJ/bit for each write operation. We derive the characteristics of this emerging memory from publicly available data [20]: we will consider a hypothetical LPRAM with a single channel, 2 ranks, 16 banks, 16384 rows, 256 columns memory, where columns are constituted of 128 single-bit memory elements, for a total storage capacity of 2 GB. Accesses are performed according to the LPDDR4X protocol, at 1600 MB/s data rate, with one Column pre-fetch. A notable difference from the LPDDR4X standard is the absence of the self-refresh and targeted refresh logic, which is not necessary due to the data persistence characteristics of the PCM technology.

We consider an unencrypted implementation as a baseline. To that result, we will compare our implementation and a XEX-based tweaked-codebook mode with ciphertext stealing (XTS) encryption based on the AES block cipher. The XTS mode of operation is the de-facto standard for disk encryption and acts as a reference of a full data encryption algorithm, although lacking address and commands encryption. The XTS algorithm is fed with the physical address of each block (to hide spatial patterns) combined with a large write counter (to hide temporal patterns).

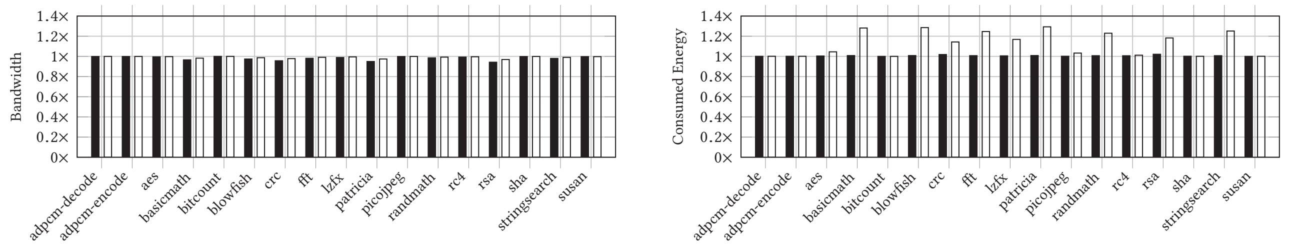
The results of our evaluation are depicted in Figure 6, our system has an average penalty of 1.8% on the memory bandwidth, which is slightly higher than the 0.9% of the AES-XTS based solution. This is the result of a slightly higher latency in our secure architecture, due to the fact that we aggregate the two DDR `DQ` values into one plaintext. On the energy consumption side we clearly benefit from saving data in plaintext form, as we achieve negligible energy overheads, while AES-XTS based solution requires 12% more energy. Cipher energy consumption is not simulated for simplicity, according to the Kumar et al. [15] implementation power metrics, the two Acorn ciphers in our implementation would consume roughly the 3.3% of the memory power. To provide a comparison, a state of the art AES-XTS ASIC implementation from Hämäläinen et al. [9] would consume 7% of the whole memory power.

The high variance visible among the various benchmarks in the Energy plot is due to the difference among read and write counts in the access patterns. Since we feed the XTS algorithm with a large counter to hide temporal patterns, every encrypted write is different from the original encrypted data, even if the data block was not changed. In particular, due to the diffusion property, on average half of the bits will be flipped for each re-write. Conversely, our implementation never stores encrypted data into memory, therefore is always aligned with the unencrypted baseline.

5 RELATED WORK

The state of the art describes two different approaches on securing Non-Volatile Main Memories. The first approach assumes that only data-remanence attacks are possible, explained in i-NVMM [6], which fully encrypt data only when the system is powered down. A second approach considers also bus snooping and tampering as an attack model, followed by ASSURE [19]. ASSURE and subsequent works employ an AES Counter (CTR) implementation for the ability to hide the encryption cost behind the memory latency.

To prevent data tampering, an HMAC algorithm is applied over the encrypted data, saving the resulting tags inside the NVM, in encrypted form. There is no need to keep the IVs used for CTR mode secret and therefore they are not encrypted. A Merkle Tree based structure is used to prevent tampering, with the root nodes stored inside the SoC Memory Controller. This approach has the drawback of having a large storage overhead, and of storing high-entropy encrypted data, which as explained in Section 2.1 degrades memory lifetime and energy figures. Furthermore, computing the Merkle Tree hashes has a non-negligible cost in terms of energy, which is logarithmic with respect to the storage size. Later work such as DEUCE [24] reduces the counters granularity (one for each byte and one for each cache lines, combined), to avoid unnecessary re-encryptions and counter reuse problems. The state of the art



(a) Bandwidth **(b) Energy**

Figure 6: Results from the simulation of the MiBench2 benchmark suite on the gem5 full system simulator, with the nvmain simulator. Black bars: our solution; white bars: AES-XTS implementation; results normalized w.r.t unencrypted baseline.

implementation carries a lot of performance drawbacks, even the best implementation of ASSURE still incurs in a write multiplication factor of 2.75x and an Instruction Per Cycle (IPC) reduction of 0.72x.

Another work is more similar to our solution: Obfusmem [1] by Awad et. al. tries to obtain the obfuscation of data, command and addresses by issuing fake memory requests encrypting addresses and data with CTR-mode encryption. Obfusmem however lies under our same attack scenarios, but its implementation lacks data integrity and performs two encryptions of the data written to memory. This destroys the advantage of storing low-entropy data into the Emerging Memory (see Sec. 2.1).

6 CONCLUDING REMARKS

We proposed a secure LPDDR4X adapter, designed to be easily integrated in future NVM-based embedded devices. As shown by our experimental evaluation, our design is able to keep up with the elevate throughput of the LPDDR4X protocol, only adding a 2% penalty vs the unencrypted baseline. From the energy perspective we fully exploit the DCW and DI implementation of emerging memories, obtaining (for the considered PCM-like memory) a zero-overhead energy profile, while the energy cost of our ciphers is just 3% of the memory energy consumption. The designed architecture, even with such small costs, guarantees confidentiality, integrity and authentication altoghether on data, commands and addresses.

REFERENCES

[1] Amro Awad, Yipeng Wang, Deborah Shands, and Yan Solihin. 2017. ObfusMem: A Low-Overhead Access Obfuscation for Trusted Memories. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017*. ACM, Toronto, ON, Canada, June 24-28, 2017, 107–119. http://doi.acm.org/10.1145/3079856.3080230

[2] Daniel J. Bernstein. 2018. CAESAR submissions. https://competitions.cr.yp.to/caesar-submissions.html

[3] Nathan L. Binkert and et al. 2011. The gem5 simulator. *SIGARCH Computer Architecture News* 39, 2 (2011), 1–7. https://doi.org/10.1145/2024716.2024718

[4] Eric Brier, David Naccache, Phong Q. Nguyen, and Mehdi Tibouchi. 2011. Modulus fault attacks against RSA-CRT signatures. *J. Cryptographic Engineering* 1, 3 (2011), 243–253. https://doi.org/10.1007/s13389-011-0015-x

[5] Cadence. 2018. Simplify DDR-PHY. http://www.ddr-phy.org/

[6] Siddhartha Chhabra and Yan Solihin. 2011. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011*, Ravi Iyer, Qing Yang, and Antonio González (Eds.). ACM, San Jose, CA, USA, 177–188. http://doi.acm.org/10.1145/2000064.2000086

[7] Shay Gueron. 2016. Attacks on Encrypted Memory and Constructions for Memory Protection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016*. IEEE Computer Society, Santa Barbara, CA, USA, 1–3. https://doi.org/10.1109/FDTC.2016.20

[8] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2009. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM* 52, 5 (2009), 91–98. https://doi.org/10.1145/1506409.1506429

[9] Panu Hämäläinen, Timo Alho, Marko Hännikäinen, and Timo D. Hämäläinen. 2006. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *Euromicro Conf. on Digital System Design: Architectures, Methods and Tools (DSD 2006), 30 August - 1 September 2006*. IEEE Computer Society, Dubrovnik, Croatia, 577–583. https://doi.org/10.1109/DSD.2006.40

[10] Matthew Hicks. 2018. MiBench ported for IoT devices. https://github.com/impedimentToProgress/MiBench2

[11] JEDEC Solid State Technology Association. 2017. ADDENDUM No. 1 to JESD209-4, LOW POWER DOUBLE DATA RATE 4X (LPDDR4X). https://www.jedec.org/standards-documents/docs/jesd209-4b

[12] JEDEC Solid State Technology Association. 2017. JEDEC Updates Standards for Low Power Memory Devices. https://www.jedec.org/news/pressreleases/jedec-updates-standards-low-power-memory-devices-0

[13] JEDEC Solid State Technology Association. 2017. LOW POWER DOUBLE DATA RATE 4 (LPDDR4). https://www.jedec.org/standards-documents/docs/jesd209-4b

[14] Nolen Johnson. 2018. Qualcomm's Chain of Trust. https://lineageos.org/engineering/Qualcomm-Firmware/

[15] Sachin Kumar, Jawad Haj-Yihia, Mustafa Khairallah, and Anupam Chattopadhyay. 2017. A Comprehensive Performance Analysis of Hardware Implementations of CAESAR Candidates. *IACR Cryptology ePrint Archive* 2017 (2017), 1261. http://eprint.iacr.org/2017/1261

[16] Micron. 2018. Breakthrough Nonvolatile Memory Technology. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology

[17] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006 (Lecture Notes in Computer Science)*, David Pointcheval (Ed.), Vol. 3860. Springer, San Jose, CA, USA, 1–20. https://doi.org/10.1007/11605805_1

[18] Matthew Poremba and Yuan Xie. 2012. NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2012*. IEEE Computer Society, Amherst, MA, USA, 392–397. https://doi.org/10.1109/ISVLSI.2012.82

[19] Joydeep Rakshit and Kartik Mohanram. 2017. ASSURE: Authentication Scheme for SecURE Energy Efficient Non-Volatile Memories. In *Proceedings of the 54th Annual Design Automation Conference, DAC 2017*. ACM, Austin, TX, USA, 11:1–11:6. http://doi.acm.org/10.1145/3061639.3062205

[20] A. Redaelli. 2017. *Phase Change Memory: Device Physics, Reliability and Applications*. Springer International Publishing, Vimercate, Italy. https://books.google.it/books?id=VIl_DwAAQBAJ

[21] STMicroelectronics. 2018. STMicroelectronics Now Sampling Embedded PCM for Automotive Microcontrollers. https://www.st.com/content/st_com/en/about/media-center/press-item.html/t4119.html

[22] Keysight Technologies. 2018. Keysight W6600A Series LPDDR4 BGA Interposers. https://literature.cdn.keysight.com/litweb/pdf/5992-1461EN.pdf?id=2723952

[23] Hongjun Wu. 2016. ACORN: A Lightweight Authenticated Cipher (v3). https://competitions.cr.yp.to/round3/acornv3.pdf

[24] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. DEUCE: Write-Efficient Encryption for Non-Volatile Memories. In *Proc. Int.l Conf. on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15*, Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas (Eds.). ACM, Istanbul, Turkey, 33–44. http://doi.acm.org/10.1145/2694344.2694387

Received September 8, 2021, accepted September 20, 2021, date of publication November 2, 2021,
date of current version December 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3124761

Exploring Cortex-M Microarchitectural Side Channel Information Leakage

ALESSANDRO BARENGHI, LUCA BREVEGLIERI, NICCOLÒ IZZO, AND GERARDO PELOSI, (Member, IEEE)
Department of Electronics, Information and Bioengineering, Politecnico di Milano, 20133 Milan, Italy
Corresponding author: Niccolò Izzo (niccolo.izzo@polimi.it)

ABSTRACT The growing Internet of Things (IoT) market demands side-channel attack resistant, efficient, cryptographic implementations. Such implementations, however, are microarchitecture-specific, and cannot be implemented without an in-depth structural knowledge of the CPU and memory information leakage patterns; a description of such information leakages is presently not disclosed by any processor design company. In this work we propose the first Instruction Set Architecture (ISA) level framework for microarchitectural leakage characterization. Our framework allows to extract a microarchitectural leakage profile from a superscalar in-order processor; we infer detailed pipeline characteristics through the observation of instruction execution timings, and provide an identification of the datapath registers via a side-channel measuring setup. The extracted model can serve as a foundation for building solid countermeasures against side-channel attacks on software cryptographic implementations. We validate the extracted models on the ARM Cortex-M4 and ARM Cortex-M7 CPUs, two of the most widespread ARM microcontroller cores. Finally, as a further validation of the effectiveness of our derived model, we mount a successful attack on unprotected AES implementations for each of the examined platforms.

INDEX TERMS Computer security, correlation power analysis, embedded systems security, microarchitectural reverse engineering, side channel attack countermeasures.

I. INTRODUCTION

Embedded and Internet of Things (IoT) industries use extensively cryptographic algorithms to guarantee authenticity on data retrieved from remote devices, integrity on firmware updates delivered to said devices, and confidentiality on data transmitted via radio channels. Since many of those devices are deployed in publicly accessible environments, where physical protection may be unpractical, their cryptographic implementations are a prime target for side-channel attacks [30] Side-channel attacks rely on extracting information which is unintendedly transmitted by measurable changes in a physical parameter of a computing device. Common physical parameters include the time required by the computation, the energy required to perform it, or the Electro-Magnetic (EM) and acoustic emissions of the device: such parameters are referred to as side-channels. The practicality of extracting information from a variety of side channels is witnessed by many public works; among which, as a sample, we recall Many side-channels such as timing variations [19], acoustic emissions [3], optical emissions [12], power consumption [20] and Electro-Magnetic (EM) emissions [14] have been investigated.

In this work, we will focus on power consumption and EM emission side-channels, due to their significant effectiveness. Although several hardware EM side-channel countermeasures have been developed, such as dual rail logic [22] and virtual secure circuits [37], nonetheless, due to their added cost, they are usually applied to dedicated Integrated Circuits (IC) such as secure enclaves [35], and are usually not present in general purpose Micro Controller Units (MCUs) [34]. Whenever hardware countermeasures are not in place, software protections can be employed. Three general families of countermeasures against power/EM side-channels have been proposed: i) *Masking* [9], [29], which mixes sensitive data with per-execution random values to hamper the use of correlation attacks; ii) *Hiding* [5], [24], [32], which lowers the signal to noise ratio of the side channel either by adding noise to it, or randomizing the order in which computations are made, in turn forcing the attacker to observe

The associate editor coordinating the review of this manuscript and approving it for publication was Gautam Srivastava.

unrelated information on the side channel; iii) *Morphing* [1], which dynamically changes the operations employed to compute the cryptographic primitive, while retaining semantic equivalence.

Masking countermeasures are of particular interest, as they can be proven to be completely blocking side channel attacks, given a chosen measurement model, known as *probing model*. Masking techniques operate combining sensitive values with random, per execution values known as *masks*. The most common combination is performed by adding via Boolean exclusive or $d \geq 1$ random values to the sensitive value itself, and considering the result, together with the d random values, as a redundant representation of the original sensitive value. A protected value is thus also said to be split into $d + 1$ *shares*, which, combined together via Boolean XOR, yield the original value. The masking approach requires to perform the original computation on the $d + 1$ shares of the sensitive value, taking care that the results are also expressed in the same redundant and randomized form. A masking scheme can be thus proven secure against an adversary able to obtain information via side channel on any number of shares up to d of a given sensitive value. This in turn implies that the computation on masked values should never combine together more than d shares of the same value, unless additional fresh randomness is added in the combination process, making the power consumption of the combination once again independent from the combined values. Masking scheme descriptions thus provide both the method to split a value into shares, and the methods to perform computations of arbitrary Boolean functions on share-split values.

While explicit value combinations through operations are avoided in the design of a masking scheme, providing a concrete implementation of the scheme itself which is faithful to this has proven to be challenging. Indeed, setting aside implementation errors, the reuse of shared memory components to store different shares may lead to a violation of the non-combination principle. In dedicated hardware designs, this can be avoided simply by dedicating memory components to the storage of a single share only. Software implementations on the other hand, are forced to employ shared memory resources, e.g. general purpose CPU registers, to store the shares during the computation. This constraint on memory element reuse is known to reduce, or, in critical cases, nullify the security margin of masking schemes. This security reduction, formalized in [4], observes that storing two shares of a given sensitive value in the same memory element, in consecutive clock cycles, will cause a power consumption proportional to the Hamming distance between the two shares, i.e., proportional to the Hamming weight of their xor-combination. In the simple case where $d = 1$, storing the two shares into which the value is split in the same memory element in subsequent clock cycles will transmit on the side channel the Hamming weight of the unprotected value itself. As a consequence of this behaviour, particular attention in avoiding careless architectural register reuse is suggested in [4], to the point of hand-coding the assembly-level description of an algorithm.

While controlling the implementation at an assembly-level description provides the means to a programmer to avoid architectural register reuse, a recent study has shown that significant information leakage is caused by the state transitions of microarchitectural registers [6]. Indeed inter-stage registers in a pipelined processor might leak a combination of values that are unrelated at ISA level, e.g., the result of two consecutive operations performed on orthogonal register sets. Such an addition information leakage can thus invalidate masking scheme implementations which appear to be satisfied at ISA-level, as shown in [32], where the authors detailed a microarchitectural leakage-fragile implementation.

Counteracting microarchitectural leakage can either be done with a hardware approach, e.g. designing an ad-hoc instruction set extension as in [15], or adapting the software to take into account the actual microarchitectural leakage. The second solution is the most flexible in terms of reuse of the existing hardware, but it requires knowledge of the CPU microarchitecture to a point where the microarchitectural leakage can be characterized.

A. CONTRIBUTION

In this work we tackle the problem of characterizing how the CPU and the memory subsystems of an arbitrary superscalar in-order processor leak the processed data due to state change of the memory elements contained in the CPU pipeline and load-store unit. To this end, we propose a microarchitecture-agnostic, ISA-level framework to characterize the power/EM side-channel after inferring the relevant microarchitectural details. The main observation is that the timing side channel provides significant information on the structure of a superscalar CPU, as CPU design aims at performing computation in the least possible amount of time. In lieu of employing timing information to derive the data being processed, as it is common in timing based side channel attack, we employ microbenchmarks with known data and instructions to derive the computation strategy.

Our framework, through the execution of a set of ad-hoc microbenchmarks, and the measurement of software-only parameters such as execution latency, throughput and physical phenomena (i.e., EM emissions), is able to yield a structural characterization of the side-channel features of a target processor and memory subsystem, such as the size and location of all the datapath synchronous registers, and the microarchitectural features necessary to correctly model their leakage behaviors.

We practically demonstrate the soundness of our methodology by applying it to two different processors of the ARM microcontrollers family: a Cortex-M4 and a Cortex-M7. In particular, the Cortex-M4 being a commercially widespread single-issue 4-stage CPU, and the Cortex-M7 a full dual-issue 5-stage superscalar CPU, currently the most powerful CPU of the Cortex-M series.

As an additional confirmation of our characterization work, we demonstrate the feasibility of a CPA attack over an unprotected AES implementation on both platforms, by recovering the correct key with less than 18000 traces. This work represents a first systematic attempt to derive a leakage model that, not only can be practically used to derive CPA-resistant software implementations, but also to gain in-depth knowledge about the causes of the observed leakage, contrarily to data-driven approaches [23], which do not aim at tracking down the causes of the observed leakage phenomena.

II. BACKGROUND

In this section we will start from a recap on the techniques employed to perform a power consumption based side-channel attack, then we recall the relevant microarchiectural pipeline details that will be analyzed in this paper, and sum up the state of the art in side-channel countermeasures and microarchitectural analysis.

A. SIDE-CHANNEL ATTACKS

The computation of hardware and software implementations of cryptographic algorithms, can be considered as an arbitrarily long sequence of states, where the first represents the input of the algorithm and the last, its output. In traditional Complementary Metal-Oxide-Semiconductor (CMOS) based computation, those states are stored as logic values of arrays of memory elements, which embody the sequential part of the processor. Such values accordingly drive the buses and logic lines that feed the combinatorial portion of the CPU. This feature is present in both Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA). In the transition between states, we observe an information leakage, on both the power consumption and Electro-Magnetic (EM) emissions side channels, The leakage was characterized to be a change in the power consumption and Electro-Magnetic (EM) emissions correlated with the number of single-bit memory elements changing state [11], [32]. In detail, referring to the leakage categorization by Balasch *et al.* [4], for every transition θ of the set Θ of all the transitions of the target implementation, we can define the sampled leakage function as the sum of a deterministic part $L_{\theta,d}$ and a measurement noise part N_θ: $L_\theta = L_{\theta,d} + N_\theta$. The $L_{\theta,d}$ part itself is correlated with the *transition leakage*, which leaks the values of transitions between states, while the N_θ term takes into account both measurement noise and the contribution of portions of the digital circuit which are not related to the computation at hand.

A typical side channel attack exploits the correlation between $L_{\theta,d}$ and the values involved in the transition θ, considering the transition of a portion of the computing device involving a small part of the secret key. It is possible for the attacker, for each (guessed) value of the part of the secret key, to build a prediction of the value of $L_{\theta,d}$ which would be measured from the device. These prediction must then be checked against the actual behaviour of the device, as it is observed via direct measurement. The best fitting prediction will reveal to the attacker what is the actual value of the key portion.

Since the measurements will obtain the value L_θ, which is affected by noise, statistical comparison tools are employed. In this work, we will adopt the Pearson correlation coefficient; as a consequence the side channel attack performed is denoted as Correlation Power Analysis (CPA). The choice of the Pearson correlation coefficient is backed by the work of Heuser *et al.* [18], which demonstrated that whenever the leakage model is only known to a proportional scale (i.e. $L_{\theta,d} = ax + b$ where both a and b are unknown, and x represents the value being leaked), and the noise is Gaussian, the Pearson coefficient is an optimal distinguisher. Indeed, in our case, the power consumption (and consequent EM emissions) of an memory element is expected to be proportional to the number of elements toggling; i.e. the leakage is proportional to Hamming distance value being leaked.

Operatively, a CPA proceeds as follows. The side channel leakage traces of a large number of encryptions (or decryptions) with random known inputs under the same key are collected. For all the possible values of a small key portion, and for all the cipher inputs, we model the leakage of the value transitions of a portion of the internal state, which can be fully determined by each combination of input and key portion hypothesis. The leakage corresponding to the update operation is modeled as the Hamming distance between the old and the updated value. We compute the Pearson correlation coefficient between the HD leakage model and the captured trace, for each different key portion hypothesis-input pairs. The correct key portion hypothesis will then be the one whose Pearson correlation coefficient value is the highest.

B. STRUCTURE OF A PIPELINED CPU

The computation of a CPU can be logically split in five different execution phases, which in a pipelined architecture become the *stages* of the pipeline itself. The Instruction Fetch (IF) stage loads one or more binary words from the instruction memory (or from the instruction cache if present), and passes them to the Instruction Decode (ID) stage. The ID stage translates each binary opcode into a combination of control signals, which activates the CPU functional units necessary to perform the encoded operation. The ISsue (IS) stage loads the registers from the Register File (RF), checks for eventual hazards and inserts pipeline wait cycles (also called stalls, or bubbles) to solve them. Finally the EXecute (EX) stage performs the actual computation and writes back the computed data to the RF. This last write back operation can be delegated to a fifth optional stage, called WriteBack (WB) [17].

A pipelined architecture needs to store the results of each stage, so that the next stage can read them in the following clock cycle; therefore there exist inter-stage registers between each consecutive pair of stages [17]. We refer to such registers with the names of their adjacent stages, for example an inter-stage register between ISsue and EXecute stages is defined IS/EX. Advanced CPUs might be equipped with

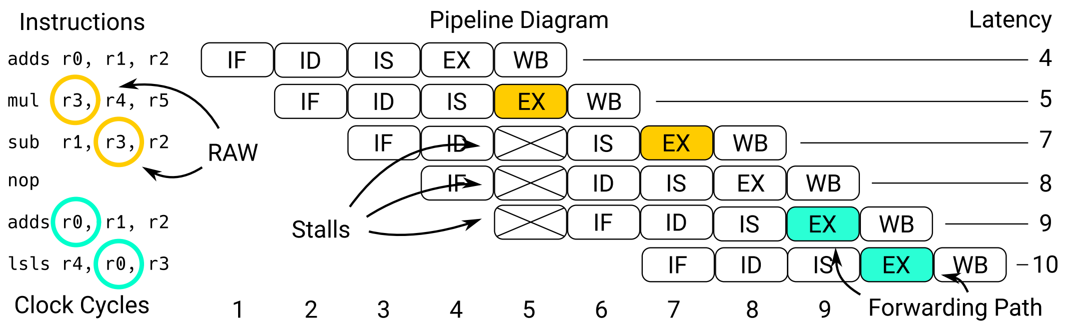


FIGURE 1. **Pipeline diagram of a 5-stage in-order CPU executing an instruction sequence reported on the left. The orange highlighting singles out the instructions causing a read-after-write (RAW) conflict, solved by inserting a stall at cycle 5. The teal coloured operations represent a case where a the presence of a forwarding path allows to avoid the insertion of the stall.**

several Issue units, which makes them *multiple-issue*, thus able to issue several instructions concurrently [17]. When analyzing a CPU microarchitecture, we need a detailed view of its execution stage, thus we need to distinguish all the functional units that act during this stage. Among them, we have one or more Arithmetic Logic Units (ALU), which are in charge of executing all the algebraic, bitwise and logic manipulations, with some of them equipped with a shifter. The employed shifters usually execute in a single cycle and are built from a matrix of parallel 2×1 multiplexers, called *barrel shifter* [21]. Other relevant functional units are those devoted to memory operations, called Load Unit and Store Unit, together they read and write to a hidden register called Memory Data Register (MDR), which is typically not directly addressable but is implicitly used by all memory operations. Multiplications are usually performed by dedicated multipliers, which will also be analyzed. Analyses concerning floating point ALUs, vector processing and the analysis of other specialized units are left to future explorations due to their comparatively minor role in implementing cryptographic schemes.

Two kinds of conflicts might occur during the execution of a processor, which are named Read-After-Write (RAW) and structural conflicts [17]. RAW conflicts take place whenever a source operand of an instruction is the destination operand of the previous one. Structural conflicts, instead, occur whenever a needed functional unit is still busy with an ongoing operation, for example in dual-issue architectures when there are single execution units in sequences of identical operations. If no such conflicts occur, e.g., if the two instructions are completely independent from one another and they use different execution units, they usually can be issued together. Instead, if a RAW hazard is present, the dependent instruction should be stalled [17], as it happens in the following ARM assembler snippet, where two instructions cannot be dual-issued because they contain a RAW hazard:

```
DIV r0, r1, r2
ADD r3, r0, r0
```

Each conflict is solved through the insertion of *stalls* [17], which are cycles in which some of the pipeline stages are stopped to let the blocking operation complete its transit through the pipeline. Regarding RAW conflicts, another way to solve them is through the use of forwarding paths [17], which are connections between inter-stage registers that allow to forward output values without storing and retrieving them from the RF. Instead, to limit the insurgence of structural hazards, CPU designers usually add replicas of the various functional units; this is especially beneficial in multiple-issue processors. To maximize their power efficiency, CPUs are designed to run at their maximum capabilities. Therefore, it is reasonable to expect that, whenever no structural or data conflicts take place, the CPU computes all the instructions, multiple-issuing them in the best, i.e., lowest, number of Cycles per Instruction (CPI) [28]. A direct consequence of this fact is that CPI values are a source of information on the inner architecture of a CPU; in particular a decrease in the CPI reveals a conflict. A sample pipelined execution is depicted in Fig. 1, where we can observe from the figure that a RAW data dependency exists between the second and the third instructions, which is resolved by adding a stall on the fifth clock cycle. Furthermore, we can notice that a forwarding path between the fifth and sixth instruction avoids the insertion of an additional stall cycle at clock cycle 10. One additional optimization that can be found in more complex CPUs, is the adoption of pipelined execution units. This optimization is applied on multi-cycle execution units, typically Multiply-And-Accumulate (MAC) or Load/Store Units (LSU). A pipelined unit applies pipelining internally, thus is internally divided into *stages*, one for each execution cycle, and whenever an ongoing computation is passed to the next stage, the current one is able to start a new computation. This approach is used to maintain a sustained throughput of 1 CPI when executing a sequence of multi-cycle operations.

C. EFFECTS OF MICROARCHITECTURAL LEAKAGE ON MASKING PROTECTIONS

Hidden microarchitectural registers pose a threat to software ciphers employing masking CPA countermeasures, as mentioned in Section I, as they can leak a value which is correlated to multiple shares, thus violating the assumption of the masking schemes, as demonstrated by Seuschek *et al.* [32].

We provide a demonstration of the aforementioned case through two running examples of how a microarchitectural leakage on the registers of a XOR operation can be used to compute the original unmasked value, de-facto invalidating the countermeasure. Given two values a and b, each represented using a first order masking scheme, a is split into two shares by using a random value x: $a = (a_0 \oplus a_1), a_0 = (a \oplus x), a_1 = x$, and b is split into two shares by using a random value y: $b = (b_0 \oplus b_1), b_0 = (b \oplus y), b_1 = y$. Their masked XOR operation $c = a \oplus b$ can be computed as $c_0 = a_0 \oplus b_0$ and $c_1 = a_1 \oplus b_1$. The splitting of values into shares can be implemented using the following assembly snippet:

```
XOR a_0, a, x
XOR b_0, b, y
```

If an undisclosed microarchitectural register leaks the linear combination of the subsequent values assigned to the first source register, we would have an inadvertent leakage of the result $c = a \oplus b$. Instead, consider a second assembly snippet, which implements the masked XOR operation that was just described:

```
XOR c_0, a_0, b_0
XOR c_1, a_1, b_1
```

If additionally the microarchitecture leaks the linear combination of subsequent values assigned to the destination registers, we would see a leakage correlated with the unmasked XOR value c as: $c_0 \oplus c_1 = (a_0 \oplus b_0) \oplus (a_1 \oplus b_1) = a \oplus b = c$.

D. RELATED WORK

Since the first pioneering work by Kocher [19], the state of the art in side-channel attack analysis and countermeasures has evolved into different approaches; many of them relying on the creation of leakage models, which can be either opaque, such as the ones generated by machine learning approaches, or explainable, as are the models generated from physical simulation and architectural reverse-engineering. Opaque models can be useful to shuffle the code until a reduced-leakage implementation is found. Instead, explainable models could be used to extend compilers to generate protected code. Orthogonally, the platforms on which such techniques are applied range from being completely known (white-box), like RISC-V compliant open-hardware processors, to being opaque (black-box), like commercial CPUs.

A first notable work to be mentioned, is set in a complete white-box scenario where the CPU is known down to the gate-level description; Sehatbakhsh *et al.*, with their EMSim tool [31], start from a gate-level model of a custom designed RISC-V processor and build a synthetic EM leakage trace generator, able of approximating the real traces of the same CPU, synthesized on an FPGA. Their methodology allows interesting developments, such as software-only verification of leakage in cryptographic algorithms or in the device validation. However such a tool is applicable only when a gate-level description of the target CPU is available, which is not the case of most commercial proprietary processors, like the two we analyse in this work. When access to a CPU gate-level description is available, further countermeasures can be directly embedded into the cores.

Gao *et al.* with their FENL [15] approach, propose an Instruction Set Extension to expose some low-level features of the microarchitecture to the compiler or even to the programmer. For example, they introduce the ability of flushing the pipeline, thus enabling compilers to isolate the computation of the shares from one another by issuing a pipeline flush in between them. Although interesting to consider for future architectures, this is still not applicable to the presently available commercial processors.

Whenever a gate-level description is unavailable, a data-driven approach can be applied, employing a machine learning model to simulate the leakage characteristics of the target architecture. McCann et. al. through their ELMO [23] framework, were able to model the complete microarchitectural leakage using machine learning tools; Shelton et. al., with the ROSITA [33] extensions to ELMO, bring the effort forward, by introducing automatic static rewriting of the cryptographic code to eliminate leakage. Both work however target a simple CPU: the ARM Cortex-M0, for which a detailed microarchitectural model is publicly available [13]. This makes both tools hardly portable to other CPUs whose microarchitecture is unknown. Furthermore, conversely to their data-driven approach, our work aims at producing an explainable model, helpful in assisting security engineers in a transparent and informed way, by overcoming the limitations of randomly changing the code until it stops leaking.

As opposed to a data-driven model, a microarchitectural approach allows for a more structured countermeasure design. Bronchain and Standaert [7] state the necessity of relying on higher-order masking implementations for protecting software ciphers on COTS hardware, and highlight the importance of investigating the actual security level which can be reached by using low-order masking. This security level can be inspected only through a thorough microarchitectural characterization. Furthermore they perform a successful attack on a masked and shuffled implementation published by Agencie Nationale de la Sécurité des Systèmes d'Information (ANSSI) in less than 2000 measurements, furtherly demonstrating the difficulty of implementing secure side-channel countermeasures.

Seuschek *et al.* [32] confirm the importance of considering the characteristics of the underlying architecture. In fact they show how a masked implementation that does not consider the microarchitectural transition leakage is ineffective against an attacker with a detailed knowledge of the architecture. They provide microarchitectural-aware instruction scheduling and register allocation passes, which try to generate code which emits a reduced leakage, upon execution on the analyzed processor. Their leakage model however is still tailored to the Cortex-M0 and is generated through manual inspection. Thus porting their implementation to a different, eventually more complex CPU, would require a complete manual inspection effort, in some cases simply not possible due to the lack of microarchitectural knowledge.

Chen *et al.* [10] describe what features should be included in a microarchitectural model. They start describing the components of the processor datapath that can generate side-channel leakage: a path from the RF to the Execution Unit, and back into the RF, used for all the arithmetic operations, bitwise and logic manipulations; a second one from the RF to the memory subsystem, used for store operations; and a last one from the memory subsystem to the RF used for loads. Chen et. al then design a custom architecture, that for certain operations, could apply a hiding countermeasure. We strive instead to build a solution for existing COTS platforms.

Barenghi and Pelosi [6] made further progress on the microarchitectural side-channel exploration path, discovering that other features of the CPU are relevant on the side-channels perspective, encompassing the RF read ports, the IS/EX inter-stage register, an eventual inter-stage register between the EXecute, and the WriteBack to the RF (EX/WB), the ALU output registers, the barrel shifter output registers, and the MDR. Finally they consider the LSU to be equipped with eventual registers used to perform eventual sub-word realignments. They apply a CPI-based characterization technique, which will be formalized and expanded in this work.

Finally, whenever an explainable model has been derived, conversely to the data-driven ones, it enables the informed design of software countermeasures. As Agosta *et al.* summarized in their work [1], different compiler-based techniques can be applied to the source code of a cryptographic implementation. A compiler with a detailed knowledge of the leakage of an architecture, such as the knowledge we are deriving in this work, will be able to directly schedule the code in a protected way, without resorting to a manual, thus long and error-prone hand-coding process. Compiler-generated countermeasures could help in protecting cryptographic code, implementing software countermeasures on each of the diverse architectures that are used today in the embedded ecosystem.

III. MICROARCHITECTURAL MODEL EXTRACTION

In this section, we describe our method to obtain a microarchitectural model of the side channel relevant features of an in-order CPU. To this end, we consider the portion of the CPU performing data-dependent activities, i.e., the CPU datapath. Providing a model of the CPU datapath useful to design software side channel countermeasures requires to model the behavior of the synchronous components in the datapath itself.

We consider the case of an in-order CPU, having, as synchronous components of its datapath: *i)* Register File (RF), *ii)* the inter-stage pipeline registers, *iii)* the Memory Address Register (MAR) and Memory Data Register (MDR) contained in the Load-Store Unit (LSU), *iv)* the store buffer, and, *v)* the inter-stage registers of pipelined execution units, if such units are present.

To derive the behavior of the synchronous components of the datapath, we classify all the microarchitectural design choices impacting on them in three categories: *i) data storage features*, including the number of register file read and write ports, *rp* and *wp* respectively, *ii) instruction issuing features*, including the issue width w, the issuing policy, the width of the inter-stage registers, and the presence of forwarding paths, and *iii) instruction execution features* including the number of available execution units, their latencies, and whether they are internally pipelined or not.

More in detail, RF updates are determined by the *data storage* microarchitectural features, since the number of RF ports shapes the updates to the Register File; pipeline inter-stage registers updates are determined by the *instruction issuing* characteristics, such as the issue width w and the issuing policy, which is non-trivial for architectures with $w > 1$, and forwarding paths, which are able to update an inter-stage register with the content of a following inter-stage register. Updates to the LSU synchronous registers, the store buffer and execution unit inter-stage registers, are all determined by the ISA semantics and by the *instruction execution* microarchitectural features. In particular, the presence of unit inter-stage registers is implied by the presence of pipelined execution units. The observation of resource contention allows to infer sharing of synchronous datapath registers among different functional units, which can cause unforeseen transition leakage.

In this work, we assume the target CPU ISA and the number of stages of the pipeline are known, as it is the case with the entire line of ARM Cortex-M CPU designs. For the sake of description clarity, as a case study, we consider a 5-stage pipeline design: Instruction Fetch (IF), Instruction Decode (ID), ISsue (IS), EXecute (EX), WriteBack (WB). If the pipeline length is unknown, an exhaustive search is always within practical reach, since the longest pipeline in a commercial CPU has 31 stages (Intel Pentium 4 Prescott), and architecture independent works demonstrate how shorter pipelines are optimal on modern workloads [16].

A summary of the approach which we will describe in the following is depicted in Figure 2. We assume the target architecture is equipped with the means of count the clock cycles taken by the execution of an arbitrary instruction sequence. This is typically available in the form of a memory mapped, clock cycle counter register which can be read at the user's will. We note that, if this feature is not available, any other means of precisely measuring the number of cycles taken by the execution of an instruction sequence (e.g., asserting and deasserting fast GPIO pins) can be used. We also assume either the knowledge, or an educated guess, on the architecture issue width w. In particular, our approach allows to validate the value of the issue width w before the rest of the microarchitectural inferences are performed.

Once the microarchitectural inferences are complete, we employ a microbenchmark suite to validate that a side

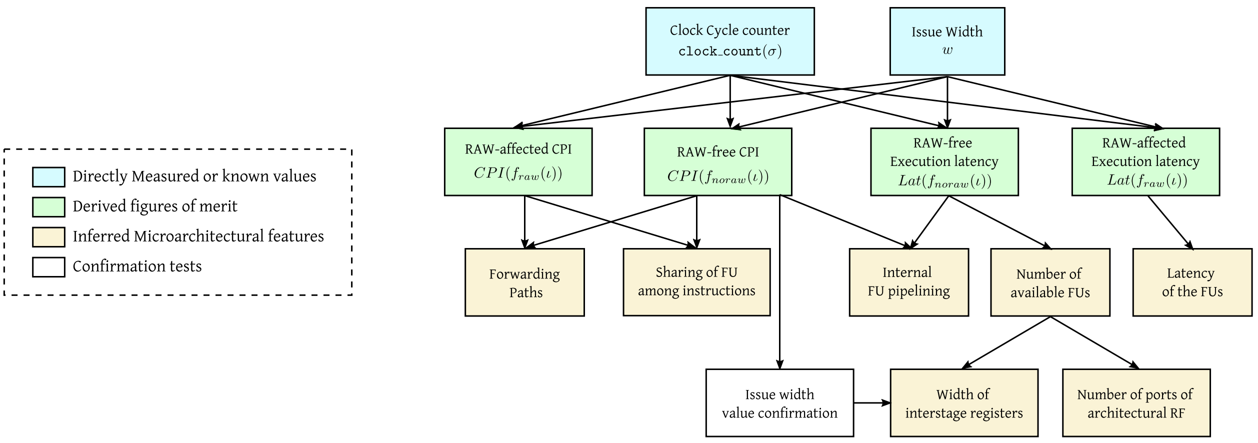

Directly Measured or known values
Derived figures of merit
Inferred Microarchitectural features
Confirmation tests
Clock Cycle counter clock_count(σ)
Issue Width w
RAW-affected CPI CPI(f_raw(ι))
RAW-free CPI CPI(f_noraw(ι))
RAW-free Execution latency Lat(f_noraw(ι))
RAW-affected Execution latency Lat(f_raw(ι))
Forwarding Paths
Sharing of FU among instructions
Internal FU pipelining
Number of available FUs
Latency of the FUs
Issue width value confirmation
Width of interstage registers
Number of ports of architectural RF

FIGURE 2. Dependency graph showing the proposed flow for deriving all the elicited microarchitectural features. An arrow pointing from one feature to the other implies that the former is used to compute the latter. The numbering of the blocks represents the topological ordering that was followed in this work to extract all the microarchitectural features.

channel information leakage matching the sequential elements we individuate is actually present.

We note that our approach, with its black-box nature, can yield a coherent description of the EM leakage, fitting well the scenario of commercial CPUs where the precise design specifications are not available. If, by contrast, the HDL description of the CPU is available a method to backtrack the side channel information leakage up to the single logic lines was described by Buhan *et al.* [8].

A. DEFINITIONS

To infer a microarchitectural model we will measure the CPU performance during the execution of tuples of instructions taken from its ISA, employed as microbenchmarks. However, performing an exhaustive exploration of all the instruction sequences that can fill up the pipeline would prove extremely time consuming. To perform a more efficient analysis, we build our microbenchmarks out of a subset **I** of the ISA, containing *representative instructions*.

Definition 1 (Representative Instruction Set, and Representative Instruction Sequence): Let **I** be a subset of the ISA, chosen to have one instruction for each functional unit of the processor. We define a representative instruction sequence ι as an arbitrary length sequence of representative instructions $\iota = \langle i_0, \ldots, i_{n-1} \rangle$ with $n \in \mathbb{N}$, where $\forall j \in 0, \ldots n, i_j \in \mathbf{I}$. We denote as $\mathbf{I}^l$ is the set of all the length-l instruction sequences, and as $\mathbf{I}^*$ is set of all the arbitrary length instruction sequences.

We expect all the instructions in the same functional unit to exhibit a similar side-channel behavior, as confirmed for the RISC-V architecture by Sehatbakhsh *et al.* [31]. To this end, we pick a single representative out of the following instruction families: no-operation, move instructions, signed addition with and without immediate, addition with shifting, multiplication, logical shift, load and store operations. Floating point instructions, which can be characterized by employing the same approach, are out of scope for this work, due to their relatively lower occurrence in cryptographic software.

Before a representative instruction sequence ι can be employed as a microbenchmark, its elements need to be specialized to act on actual data registers. Denoting the available general purpose registers as integers in the set $\{0, \ldots, \texttt{max\_reg}\}$, and assuming for the sake of simplicity that all general purpose registers can be used both as source and as destination of an instruction, we define the set of specialized instructions as follows:

Definition 2 (Specialized Instruction Set, and Specialized Instruction Sequence): Let $i \in \mathbf{I}$ be an instruction, we denote as $\texttt{src\_reg}_i$ the amount of its source registers and as $\texttt{dst\_reg}_i$ the amount of its destination registers. The destination registers and source registers of an instruction can be represented as tuples of register indices. The set of specialized instructions **S** is the set of all the tuples $s = \langle i, dr_s, sr_s \rangle$, having $i \in \mathbf{I}$, $dr_s \in \{0, \ldots, \texttt{max\_reg}\}^{\texttt{dst\_reg}_i}$, $sr_s \in \{0, \ldots, \texttt{max\_reg}\}^{\texttt{src\_reg}_i}$.

We follow the same convention adopted for the representative instruction sequences, denoting as $\mathbf{S}^*$ the set of all the arbitrarily long sequences of specialized instructions $\sigma \in \mathbf{S}^*$, where $\sigma = \langle s_0, \ldots, s_n \rangle, n \in \mathbb{N}, s_i \in \mathbf{S}$. Similarly, we denote as $\mathbf{S}^l$, $l \in \mathbb{N}$, the set of all the length-l sequences of specialized instructions.

Both in the case of representative instruction sequences and specialized instruction sequences, we indicate with the juxtaposition of the symbols, the concatenation of two sequences, e.g., $\sigma_0\sigma_1$ denotes the concatenation of the instruction sequences σ_0 and σ_1.

Translating a representative instruction sequence into a specialized one will determine whether or not most structural pipeline conflicts take place. Among the possible causes, one is easily controlled during the microbenchmark design, that is, the Read-After-Write (RAW) conflicts. A RAW conflict takes place whenever, in a specialized instruction sequence, an instruction has among its source registers at least one of the destination registers of a previous instruction. To this end, we define a logical predicate $\mathrm{RAW}(\cdot, \cdot) : \mathbf{S} \times \mathbf{S} \rightarrow \{\texttt{true}, \texttt{false}\}$ over two specialized instructions, which is true if and only if the two instructions have a RAW conflict, considering the second specialized instruction as executed after the first. We thus have that the RAW predicate is true on two specialized instructions s, r if and only if $\text{RAW}(s, r) := dr_s \cap sr_r \neq \emptyset$.

An *instruction specialization function* $f(\cdot)$ is a function defined over the free monoid of representative instructions, which maps instruction sequences to the specialized instruction sequences. Let $\iota = \langle i_0, \dots, i_l \rangle, \iota \in \mathbf{I}^*$ be a representative instruction sequence, i_j will be its j-th instruction; function $f(\cdot)$ will map ι to a specialized instruction sequence $\sigma \in \mathbf{S}^*$,i.e., an instruction sequences acting on defined architectural registers, where each member $s_j = \langle i_j, \cdot, \cdot \rangle$ is a specialization of the corresponding instruction i_j. To model the architectural behaviour in case of presence or absence of RAW conflicts, we define two different instruction specialization functions.

Definition 3 (Instruction Specialization Functions): $f_{raw}(\cdot) : \mathbf{I}^* \rightarrow \mathbf{S}^*$ is an instruction specialization function, defined over the free monoid of representative instructions, which maps each representative instruction sequence to a specialized sequence, such that the RAW predicate holds on each pair of consecutive instructions. $f_{noraw}(\cdot) : \mathbf{I}^* \rightarrow \mathbf{S}^*$ is an instruction specialization function, which maps to every instruction sequence, a specialized instruction sequence in which the RAW predicate never holds when evaluated on ordered instruction pairs.

To impose on each sequence of specialized instructions $\sigma = \langle s_0, \dots, s_j \rangle$ a RAW conflict, every member of the tuple $s_j = \langle i, dr_s, sr_s \rangle, i \in \mathbf{I}, dr_s \in \{0, \dots, r_{max}\}^{dr_{max}}, sr_s \in \{0, \dots, r_{max}\}^{sr_{max}}$ must have at least one output register operand $dr_{max} > 0$, and at least one input register operand $sr_{max} > 0$. For tuples which contain instructions where this condition does not hold, the $f_{raw}(\cdot)$ function is not defined. Where $f_{raw}(\cdot)$ is defined, we choose a set of register indices such that one of the output registers of an instruction s_j is also one of the source registers of the following instruction s_{j+1} in the sequence; the resulting register selection causes a RAW conflict between all the consecutive instructions. Let $\iota \in \mathbf{I}^*$, be a representative instruction sequence, given $\sigma = f(\iota) = \langle s_0, \dots, s_l \rangle$ as its specialized version, if the RAW predicate $RAW(s_j, s_{j+1})$ holds for $j \in \{0, \dots, l-1\}$, the specialization function can be denoted as $f_{raw}(\cdot)$.

B. MEASURING LATENCY AND DERIVING CPI

In the following, we describe how we exploit the availability of a clock cycle measurement facility and the knowledge of the issue width w (cyan elements in Figure 2) to derive the values of the clock Cycles per Instruction (CPI) $\mathtt{CPI}(\sigma)$ and the execution stage latency $Lat(\sigma)$, for a given specialized instruction sequence σ (green elements in Figure 2). Our strategy is tolerant to possible errors in the known value of w, which can be operatively corrected when the values for $\mathtt{CPI}(\sigma)$ are obtained.

For the sake of explanation, we consider the case of the clock cycle counter of ARM Cortex-M microcontrollers,

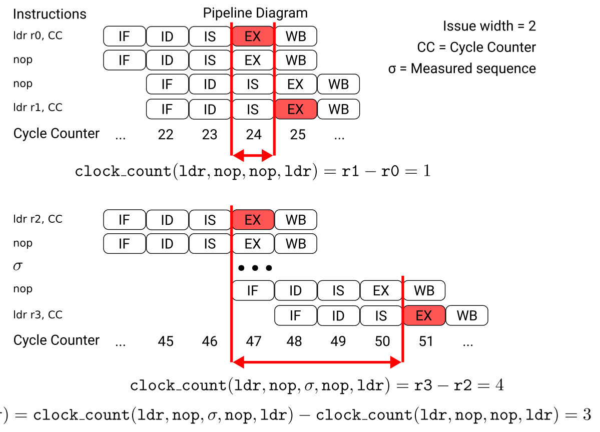

Instructions
Pipeline Diagram
Issue width = 2
CC = Cycle Counter
σ = Measured sequence
ldr r0, CC
nop
nop
ldr r1, CC
Cycle Counter
IF ID IS EX WB
... 22 23 24 25 ...
$\mathtt{clock\_count}(\mathtt{ldr}, \mathtt{nop}, \mathtt{nop}, \mathtt{ldr}) = \mathtt{r1} - \mathtt{r0} = 1$
ldr r2, CC
nop
σ
nop
ldr r3, CC
Cycle Counter
... 45 46 47 48 49 50 51 ...
$\mathtt{clock\_count}(\mathtt{ldr}, \mathtt{nop}, \sigma, \mathtt{nop}, \mathtt{ldr}) = \mathtt{r3} - \mathtt{r2} = 4$
$Lat(\sigma) = \mathtt{clock\_count}(\mathtt{ldr}, \mathtt{nop}, \sigma, \mathtt{nop}, \mathtt{ldr}) - \mathtt{clock\_count}(\mathtt{ldr}, \mathtt{nop}, \mathtt{nop}, \mathtt{ldr}) = 3$

FIGURE 3. Execution latency measuring process in a dual issue ARM architecture. The cycle counter is loaded from the symbolic memory location CC**, cycle intervals are computed by subtracting subsequent cycle counter values, and the execution cycles of a target instruction sequence σ are measured by subtracting two cycle intervals, one with an empty sequence in between, and another with σ in between.**

available as a memory mapped register, which can be read by means of a load (`ldr`) instruction. The net effect of the `ldr` instruction is to store in the target register the value of the clock cycle counter when the `ldr` instruction is in the EX pipeline stage.

We denote as $\mathtt{clock\_count}(\varsigma)$, with $\varsigma = \langle f_{noraw}(\mathtt{ldr}), \sigma, f_{noraw}(\mathtt{ldr}) \rangle$ the result of the difference between the clock cycle counter value read by the latter `ldr` instruction in ς and the one read in the former `ldr` instruction in ς. In single-issue architectures $\mathtt{clock\_count}(\varsigma)$ allows an easy derivation of $Lat(\sigma)$, subtracting the execution latency of the clock cycle counting instruction itself $Lat(f_{noraw}(\mathtt{ldr}))$, obtained as $\mathtt{clock\_count}(f_{noraw}(\mathtt{ldr}, \mathtt{ldr}))$.

Multiple issue architectures require a slightly more complex procedure to obtain the correct value of $Lat(\sigma)$, since the clock cycle counter readout instruction can be multiple-issued together with the first instructions composing σ. If this happens, the value of $\mathtt{clock\_count}(\varsigma)$ will differ in a non-trivial way from $Lat(\sigma)$, since the last clock cycle readout will be performed in parallel to the execution of the last instruction(s) of σ. To handle this issue, we exploit the knowledge of the issue width w of the underlying architecture, and pair the clock cycle readout instructions with $w - 1$ instructions which can be multiple issued together with it. These instructions act as a filler for the unused issue lanes, effectively placing the clock readouts in the appropriate cycle in time. As an example, we report in Figure 3 the case of a dual issue ($w = 2$) architecture. The example assumes that the clock-cycle readout instruction `ldr` takes a single cycle to be executed, and can be dual-issued with others. Note that in this case $\mathtt{clock\_count}(f_{noraw}(\mathtt{ldr}, i_0, i_1, \mathtt{ldr})$, where i_0 and i_1 are two instructions which can be dual issued with `ldr` would match $\mathtt{clock\_count}(f_{noraw}(\mathtt{ldr}, i_0, \mathtt{ldr}))$, since the second clock readout would be performed in parallel to the execution of i_1. In this case, the `ldr` instructions employed to read the clock cycle counter are padded with $w - 1 = 1$ instructions (`nop`s in the example) to fill the remaining issue lanes. We thus obtain $Lat(f_{noraw}(i_0, i_1))$ as $\texttt{clock\_count}(f_{noraw}(\texttt{ldr, nop}), \sigma, f_{noraw}(\texttt{nop, ldr}))-$ $\texttt{clock\_count}(f_{noraw}(\texttt{ldr, nop}), f_{noraw}(\texttt{nop, ldr}))$

1) MEASURING $\texttt{CPI}(\sigma)$

Given the correct value of w and $Lat(\sigma)$ computing $\texttt{CPI}(\sigma)$ can be achieved applying the definition, i.e., $\texttt{CPI}(\sigma) = \frac{Lat(\sigma)}{\texttt{length}(\sigma)}$.

We observe that, even in the case where the value of w is unknown, it is still possible to measure $\texttt{CPI}(\sigma)$, and, as a consequence obtain a sound estimate for the issue width of the architecture at hand, when considering instructions from the relevant instruction set $\mathbf{I}$. Indeed, in a generic w-issue CPU, each cycle counting instruction may be concurrently-issued with at most $w-1$ instructions of σ, thus preventing the straightforward elimination of measurements error in $Lat(\sigma)$ as when w is known. If the clock cycle counting instruction $s_{\texttt{count}}$ is issued with the first $w-1$ instructions of the sequence to be measured, the actual counter readout will be performed after the clock cycle counting instruction will be executed, by subtracting its execution latency from $\texttt{clock\_count}(\sigma)$. A symmetric condition takes place whenever the clock cycle counting istruction executed at the end of the sequence is multiple-issued with the last instructions of σ. In this case, the measured latency of the last w-wide instruction bundle where $s_{\texttt{count}}$ is issued is reduced to $Lat(s_{\texttt{count}})$. As a result, the value of $\texttt{clock\_count}(\sigma)$ may be smaller than $Lat(\sigma)$ by an amount upper limited by $s_{\texttt{count}} + (\ell - s_{\texttt{count}}) = \ell$, where ℓ is the execution latency of the slowest instruction of the architecture. Since the maximum reasonable precision for the $\texttt{CPI}(\sigma)$ figure is $\frac{1}{w}$ (as no smaller fraction can be achieved by a w-issue architecture), a reliable measure of $\texttt{CPI}(\sigma)$ can be obtained as:

$$\texttt{CPI}(\sigma) = \text{RoundNearest}\left(\frac{\texttt{clock\_count}(\sigma^{w\ell})}{w\ell \cdot \texttt{length}(\sigma)}, \frac{1}{w}\right)$$

Note that, when the issue width w is not known a priori, employing a suitable upper bound on its value, w', results in obtaining CPI values which are rounded to a fraction $\frac{1}{w'} < \frac{1}{w}$. Given this fact, it is possible to derive the correct value of the issue width, or verify the one being known, finding the smallest value taken by the CPI over all the relevant instruction sequences executed with no RAW dependencies, i.e. $\min_{\iota \in \mathbf{I}^{w'}} \texttt{CPI}(f_{noraw}(\iota))$, and derive w as:

$$w = \left\lceil \frac{1}{\min_{\iota \in \mathbf{I}^{w'}} \texttt{CPI}(f_{noraw}(\iota))} \right\rceil$$

C. MICROARCHITECTURAL FEATURE INFERENCE

We now describe how to extract the set of microarchitectural features allowing to build a side channel leakage model (orange elements in Figure 2). To this end, we need to identify which synchronous elements compose the datapath, and how their contents change during a computation.

We derive the aforementioned features by examining the values of $\texttt{CPI}(f_{noraw}(\iota))$ and $\texttt{CPI}(f_{raw}(\iota))$, for all the admissible length-w representative instruction sequences ι, and the values $Lat(f_{noraw}(\iota))$, for all the representative instructions i.

1) FORWARDING PATHS

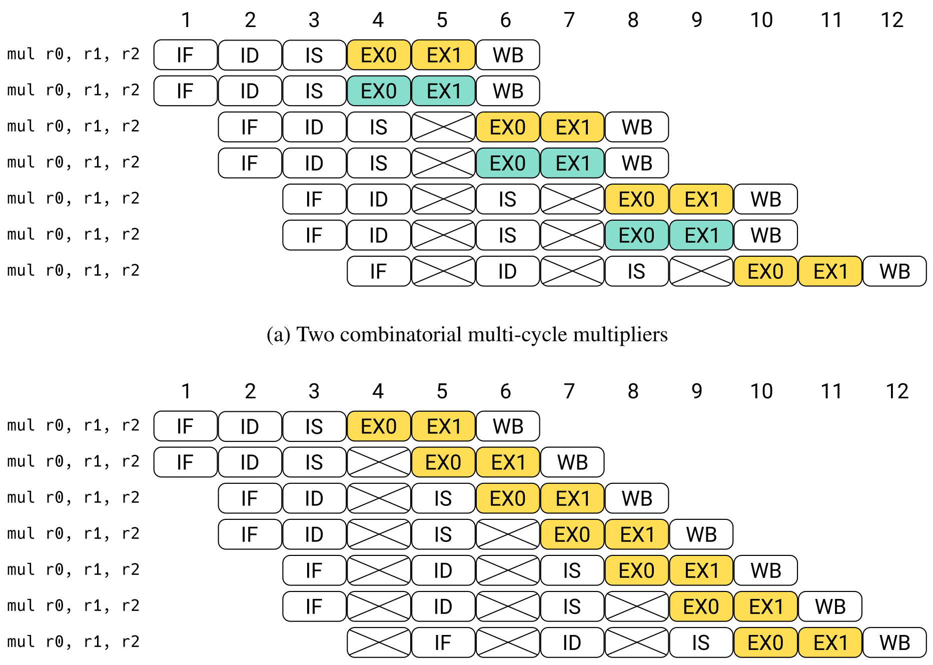
We derive the presence of a forwarding path from the output of the EX stage or the MEMory (MEM) stage to the input to the EX stage, by comparing the execution latency of $2w$ long sequences of representative instructions, constituted by the concatenation of two instances of $\iota = \langle i_0, \texttt{nop}^{w-1}\rangle$, where i_0 is either a computational or a load-store instruction, depending on the nature of the forwarding path to be investigated. The reason for inserting a $w-1$ $\texttt{nop}$ padding sequence between the two representative instructions, is to simplify the forwarding path analysis, uniforming the behavior of a multiple-issue architecture to the one of a single-issue one. Comparing the execution latency of $\sigma_{noraw} = \texttt{CPI}(f_{noraw}(\iota^2))$ and $\sigma_{raw} = \texttt{CPI}(f_{raw}(\iota^2))$, we detect the presence of a forwarding path in case the two execution latencies match; instead, in case $Lat(\sigma_{raw})$ is higher than $Lat(\sigma_{noraw})$, we detect the insertion of pipeline stalls as a result of the absence of the forwarding paths.

2) NUMBER OF AVAILABLE FUs, THEIR LATENCY AND THEIR PIPELINING

The number of functional units present for a given operation will not exceed the one which can be employed when the w issue architecture is actually issuing w concurrent instructions. Indeed, any extra FU would simply be unused, even under the best possible computation conditions, leading to a waste of resources. As a consequence, we detect the maximum number of functional units for a given operation from the execution latency of representative instruction sequences obtained from $1 \leq l \leq w$ repetitions of a single representative instruction without RAW hazards, $\sigma = f_{noraw}(i^l \texttt{nop}^{w-l})$. The padding with $\texttt{nop}$ instructions is added in order to saturate the remaining lanes of the issue unit, assuming that the $\texttt{nop}$ operation is found by the instruction issuing features investigation to be amenable to multiple issuing with any other instruction. We specialize the said representative instruction sequences, so that no RAW conflicts are present, in order to prevent data-hazards from allowing the simultaneous issuing of instructions on the available functional units. The execution latency $Lat(\sigma)$ of the sequence is expected to increase when the number of representative instruction repetitions l rises above the available functional units. Therefore, we determine the number of functional units for the said instruction as the maximum value l for which $Lat(f_{noraw}(i^l \texttt{nop}^{w-l})) = Lat(f_{noraw}(\langle i, \texttt{nop}^{w-1}\rangle))$ holds.

The latency of a given FU is simply detected as either $Lat(f_{noraw}(\langle i\rangle))$ or $Lat(f_{raw}(\langle i\rangle))$, where i is a relevant instruction computed by the FU under analysis.

In order to fully characterize the FUs themselves, beyond their latency, we should ascertain their design as either combinatorial circuits taking more than one clock cycle to stabilize, or pipelined circuits. To do so, we observe whether



(a) Two combinatorial multi-cycle multipliers

(b) Single pipelined multiplier

FIGURE 4. Pipeline timeline schemes of a dual non-pipelined and a single fully-pipelined multi-cycle multiplicative execution units. Although both implementations achieve a CPI value of 1 on a sequence of `mul` **instructions, the execution latency allows to tell the designs apart. In this scheme the dual-issue architecture is equipped with EX-EX forwarding paths.**

$\texttt{CPI}(f_{noraw}(\langle i^l \rangle))$ equals $\frac{1}{\texttt{number\_of\_FUs}}$, in which case we deem the FUs to have a pipelined design, or whether it exceeds the said value, in which case the unit design is deemed to be combinatorial. Indeed, in order to be able to sustain a CPI of $\frac{1}{\texttt{number\_of\_FUs}}$, the functional units must be designed with synchronous elements, so as to be able to accept a fresh input set at each clock cycle. Exploiting the combination of the execution latency and CPI metrics allows us to distinguish clearly between two possible cases, which would be impossible to tell apart from the CPI metric alone. As a running example, consider the two different implementation choices shown in Fig. 4 to achieve a unitary CPI on a sequence of multiplication instructions. This can be achieved with either two combinatorial multipliers, taking two clock cycles each, or a single pipelined multiplier. However, the single pipelined multiplier design will exhibit an increase in the execution latency when the number of multiplications in a sequence $\iota = \texttt{mul}^l\texttt{nop}^{2-l}$ increases from one to two, therefore allowing us to detect the presence of a single multiplier. Combining this inference with the unitary CPI, we are able to deem the unit as pipelined, as expected.

3) FU CONTENTION AMONG DIFFERENT INSTRUCTIONS

The last microarchitectural feature to be analyzed in the instruction execution feature category is the potential FU resource contention by different representative instructions. While it is usually possible to cluster representative instructions by the functional unit expected to compute them all, it is possible for different FU to share resources. The typical case is the one of a Load-Store Unit (LSU) sharing the use of the adder circuit to compute destination addresses with the Arithmetic-Logic Unit (ALU). We detect this microarchitectural resource sharing, which has the potential to cause data serialization in the pipeline interstage stage buffers as it causes a structural hazard. Therefore, considering two representative instructions i_0, i_1, belonging to different FUs, we detect a resource sharing whenever the said instructions cannot be multiple-issued together, despite the absence of RAW conflicts. This can be inferred comparing $\texttt{CPI}(f_{noraw}(\langle i_0, i_1 \rangle))$ and $\texttt{CPI}(f_{raw}(\langle i_0, i_1 \rangle))$: if no multiple issuing is possible, the two figures will match; instead if dual issuing is possible, the CPI obtained when no RAW conflicts are present will be lower.

4) NUMBER OF ARCHITECTURAL RF PORTS AND INTERSTAGE REGISTER WIDTHS

The minimum width of inter-stage registers is bound by the issue width w of the architecture and the operand requirements of the FU present in the architecture. In particular, the Issue to Execute (IS/EX) inter-stage register must be wide enough to accommodate all the data of the most demanding instruction bundle in terms of inputs. The inter-stage register between the Execute and WriteBack stages (EX/WB) the contents to be written back into the register file. As a consequence, the number of architectural words stored in each interstage register can be determined finding the maximum amount of read operands and written operands among all the possible instruction bundles issued simultaneously. We note that the maximum amount of read/written registers may be achieved in an instruction bundle with less than w instructions, in case of particularly demanding ones such as the fused multiply and accumulate instruction.

Register File (RF) ports are an expensive resource, and thus tend to be designed in a number which is the strict minimum required to exploit the FU to their utmost. We therefore determine the number of read ports and write ports of the RF by counting the maximum number architecture words which should be read from and written to the architectural register file during the computation of an instruction bundle.

D. EVALUATING THE LEAKAGE OF THE SYNCHRONOUS COMPONENTS

After describing our microarchitectural inference procedure, we now provide the means to cross-validate the inferred synchronous components and their behaviour by means of a power consumption/EM emissions side channel analysis. To do this, we turn on its head the common power consumption/EM emissions side channel attacks, where the architecture is assumed to be known, and the processed data are inferred. Indeed, through the use of carefully crafted benchmarks, where the instructions and data being processed are known, we validate the presence and behaviour of the synchronous components of the CPU.

To this end, we consider the information leakage on the power consumption and EM emissions side channels caused by the switching activity of the synchronous components. Such an information leakage is essentially proportional to the amount of single-bit synchronous memory elements switching in a given clock cycle. We therefore design microbenchmarks crafting stimuli which elicit the switching activity of a single, or a small group, of synchronous elements. We note that this can be achieved measuring the behaviour of the device in a specific time interval, and excluding the eventual instructions required to prepare the pipeline in the desired state. To detect if the side channel behaviour of the device matches the expectations suggested by a given synchronous component behaviour, we compare a sequence of predicted side channel behaviours, obtained varying the input data in a fixed instruction sequence with the actual measurements. Such an evaluation method exploits the theoretical result of modeling optimality reported in [18], where the authors state that, under the hypothesis that the component being modeled has a leakage proportional to its switching activity, employing the Pearson correlation coefficient between the toggle-count and the measured power consumption allows to extract the full information concerning how well the consumption is modeled. We now describe in detail the microbenchmarks for the specific components.

1) REGISTER FILE

To obtain a leakage behaviour related to the switching activity of the register file, we fill, outside of the side-channel measured computation, two registers with input values, and we measure the power consumption of the entire lifecycle of a `mov` instruction, through the pipeline. The said `mov` instruction is expected to generate a leakage proportional to the Hamming distance between the contents of the two registers, due to the switching activity of the write port, and the destination register. Such a leakage is expected to be present in the clock cycle when the actual write operation takes place. We note that this microbenchmark also elicits side channel leakage from the interstage registers, therefore its results should be analyzed together with them to single out the register file activity.

2) IS/EX PIPELINE REGISTER

The IS/EX inter-stage register is overwritten by the input values of the instructions issued in each slot. To single out its activity, we load $2w$ distinct inputs in the architectural register file. We then prepare the pipeline state copying the said $2w$ values into $2w$ destination registers distinct from both the source ones, and among themselves. This operation is performed so that no switching activity in the architectural register file will take place when measuring the one of the IS/EX pipeline register. We subsequently clear the interstage registers with a sequence of `nop` instructions and measure the switching activity of the pipeline re-executing the $2w$ move instructions. We employ as power model the Hamming distance of the values being stored in the IS/EX inter-stage register, that is, we compute the Hamming distance between the source operands of moves being issued in the same slot of the two w wide move instruction bundles.

3) EX/WB PIPELINE REGISTER

The EX/WB inter-stage register is updated with the destination register values of all the retired instructions. We expect this buffer to leak the HD of the computation results of instruction pairs taken from two consecutive sequences of $2w$ instructions. Our leakage testing strategy for the EX/WB interstage registers follows an approach similar to the one of the IS/EX register with a single difference. To reduce the potential unintended, and matching leakage of the IS/EX register, we employ a sequence of instructions where the result differs from the operands. In particular, in our tests, we employ additions, picking randomly both addends as our representative instructions for this test. Similarly to the strategy for the IS/EX leakage evaluation, we precharge the contents of both the source and destination registers in the RF to the inputs and outputs of $2w$ distinct addition operations. Once the pipeline state is initialized as described, we measure the power consumption of the target device during a sequence of $2w$ additions, and correlate the power consumption of the device with the Hamming distance of the results of additions being issued in the same slot as the instruction bundle.

4) INTERNAL DATA STORAGE IN THE LSU

The Memory Address Register (MAR) Memory Data Register (MDR), and the store buffer, if present, are expected to

TABLE 1. Number of source and destination 32 bit registers for the representative instruction set of the ARMv7-M Thumb ISA.

Set of Instructions	No. of Source Registers	No. of Destination Registers
`nop`	0	0
`mov,addilsli`	1	1
`ldr,str`	1	1
`add,addsh,mul,lsl`	2	1
`ldrd,strd` (load,store double word)	2	2
`smlal` (32 bit equivalent registers)	4	2

exhibit an information leakage proportional to their switching activity. As a consequence of our pipeline characterization, we are also able to validate such a fact. To this end, prepare the pipeline and memory state taking care of storing a copy of $2x$ random values both in $2x$ distinct registers, and in $2x$ memory locations, where x is the number of LSUs present in the CPU, assuming that `load`/`store` operations can be multiple-issued. Subsequently, we measure the power consumption of x load operations that fetch from the main memory the first x values, storing them in the registers that have been precharged to the said values during the pipeline preparation phase. We then execute w or more non-memory instructions, followed by another run of x load operations, loading the remaining x values onto the registers containing a copy of them, placed there in the preparation phase. This sequence of instruction is expected to exhibit a power consumption proportional to the Hamming distance between the values loaded by load instructions computed by the same LSU, regardless of the execution of purely computational instructions in between.

5) RESOURCE CONTENTION ON SYNCHRONOUS ELEMENTS

Sharing portions of the execution stage among different FUs may lead to have also shared synchronous elements. To this end, if resource sharing among FUs is detected, employing a sequence of instructions, interleaving instructions which are computed by different FUs will elicit specifically the use of the shared buffers. Since in case no resource sharing is present, no interaction among the units should happen, correlating the power consumption of the circuit during the interleaved instruction sequence with the Hamming distance between the values taken by the expected shared intermediate result (e.g. the target address of the LSU and the sum of two register values), will lead to the confirmation or denial of the presence of the shared synchronous element.

IV. EXPERIMENTAL EVALUATION

In this section we describe the adopted measurement setup, and the results of the application of the microarchitectural inference techniques exposed in Section III to two real-world in-order CPUs: the ARM Cortex-M4 and the ARM Cortex-M7. We validate the presence of the information leakage on the EM emissions side channel stemming from the synchronous components we infer as described in Section III. Finally, we compare the results of our microarchitectural inference procedure against the publicly available information on the two microarchitectures.

A. CASE STUDY MICROARCHITECTURES

ARM Cortex-M4 and ARM Cortex-M7 are two microcontroller-grade CPUs, compliant with the same ARMv7-M Thumb ISA [2]. Our first step is to define the representative instruction set for the ARMv7-M Thumb ISA, selecting a representative for each of the instruction families listed in Subsection III-A. As a result, our microbenchmarks representative instruction set **I** is composed as follows:

- No-Operation: `nop`
- Move operation: `mov rA, rB`
- Signed addition with and without immediate operands: `adds rA, rB, rC` and `adds rA, rB, #imm`, which we denote, for the sake of compactness as `add` and `addi`. Whenever their behaviour does not differ, we indicate both of them as `add[i]`
- Addition with operand shifting: `add rA, rB, rC, lsl #immediate`, denoted in short as `addsh`
- Multiplication: `mul rA, rB, rC`
- Logical Shift with and without immediate operands `lsls rA, rB, rC` `lsls rA, rB, #immediate`. Analogously to the `add` case we denote the logical shift without immediate as `lsl`, the one with an immediate as `lsli`, and any of the two instructions, as `lsl[i]`
- Load and store instructions of single 32b words: `ldr rA,[rB]`, `str rA,[rB]`
- Load and store instructions of double 32b words: `ldrd rA,[rB]`, `strd rA,[rB]`
- Signed Long Multiply, with optional Accumulate `smlal rClow, rChigh, rA, rB`.

Given the chosen representative instructions, we can compute the number of RF read and write ports required by each one of them, deriving it from their number of source registers and destination registers. This information, which is functional to the derivation of the number of read and write ports of the register file for both the Cortex-M4 and Cortex-M7 CPUs, is reported in Table 1.

B. MEASUREMENT SETUP

We chose as the Cortex-M4 and Cortex-M7 implementations for our exploration two microcontrollers produced by STMicroelectronics, namely the STM32F401 microcontroller, mounted on a Nucleo-F401RE board, for the Cortex-M4 and the STM32F746ZG, mounted on a Nucleo-F746ZG board, for the Cortex-M7.

On both microcontrollers, the `clock_count`$(\cdot)$ measurement is done employing the Data Watchpoint and Trace (DWT) debugging subsystem, which contains a clock cycle counter accessible via a single-cycle load (`ldr`) operation. Cortex-M architectures allow for the presence of both instruction and data caches, at the designer's will. In order to cope with the potential non-determinism introduced by cached load and store operations, we performed all our `clock_count`$(\cdot)$ measurements taking care of preheating the instruction cache with the specialized instruction sequence we want to measure, and the data cache with the in-memory data that are handled by the said specialized instruction sequence. Employing the pre-heating strategy, we were able to remove the effects of the variable latency coming from storing the executable code segment in Flash, as all the measured instructions are fetched from the CPU instruction cache. Additionally, we ensure that the instruction sequence to be measured can fit into a single instruction cache line, condition which can be empirically verified by observing how the variability of the execution latency decreases as the size of the measured code is reduced below a platform-dependent threshold.

The firmware that executes the described measuring algorithm is based on the Miosix kernel [36], which natively supports both boards. The Operating System (OS) is used to facilitate access to Input/Output facilities, such as logging functions and the Universal Asynchronous Receiver-Transmitter (UART) communication, but is not a prerequisite for the implementation of our framework. To zero out possible interferences of the Operating System (OS) kernel, the interrupts are kept disabled during the execution of the microbenchmarks.

The side channel measurement setup employed to validate our microarchitectural inferences is the same for both target boards. In particular, we measured the Electro-Magnetic (EM) emissions of the microcontrollers under analysis by means of a custom loop probe derived from a 50Ω coaxial cable, connected to two cascaded Agilent INA-10386 amplifiers with a gain of 26 dB each. The amplified probe is connected to a PicoScope 5244D digital sampling oscilloscope, sampling at 500Msamples/s, with an 8-bit per sample vertical resolution. We collect 200k measurements of the EM emissions over the microcontroller package, for each one of the microbenchmark sequences described in Section III. The operations of the microbenchmarks are performed on randomized inputs, obtained as the output of an AES-128-CTR based PseudoRandom Number Generator (PRNG). We correlate the measured EM emissions with the switching activity of the synchronous component whose leakage we want to evaluate, taking as the model the Hamming distance of its contents in two subsequent clock cycles. We employ the Pearson sample correlation coefficient as a measure of the correlation, deeming a sample correlation to be significant whenever its confidence interval for $\alpha = 0.1$ is disjoint from the confidence interval for null correlation, with the same significance level.

The employed CPA methodology could lead to false positives, as the hypothesized component could not be the only one leaking the expected value. However, a model containing more registers than the ones actually present, will actually lead to the implementation of a masking scheme employing more shares than necessary, but still will not violate the masking countermeasure assumptions.

C. MICROARCHITECTURAL INFERENCE ON CORTEX-M4

The first step in analyzing the Cortex-M4 pipeline was to confirm the fact that the CPU is has indeed issue width $w = 1$. To this end we measured the values of $\texttt{CPI}(\sigma)$ for instruction sequences of 200 instructions, to obtain a negligible error in the CPI measurement as specified in Section III. We observed that the minimum CPI value is achieved with a sequence of 2 instructions, and its value is 1, thus indicating a single-issue CPU. We also know from the publicly available information [25] that the Cortex-M4 CPU has a three-stages pipeline; we therefore do not need to infer the pipeline depth.

Having confirmed the issue width value, we proceeded to the measurement of the values of $\texttt{CPI}(f_{raw}(\langle\iota_0\iota_1\rangle))$ and $\texttt{CPI}(f_{raw}(\langle\iota_0\iota_1\rangle))$ for any two instruction pairs of the representative instruction set, which are reported in Table 2. For the sake of a more effective readout, we highlight in green the specialized instruction pairs attaining the minimum CPI, and in increasingly hotter colours the others.

Observing that, for all computational instructions (i.e., non-load/store instructions), the Cortex-M4 is able to maintain a CPI of 1 even in case of RAW hazards between them, we infer the presence of an EX/EX forwarding path. We note that, as the CPU is a single issue one, we deduce that at most one functional unit per operation is present in the CPU, and no sharing of functional units among different instructions takes place.

After analyzing the CPI data of Table 2 alone, we proceeded from left to right in the inferences allowed by our framework as per depiction in Figure 2. We therefore measured the value of $Lat(f_{raw}(\iota)^n)$ for $1 \leq n \leq 9$ for each instruction in our relevant instruction set. The obtained latency values are reported in Table 3: as it can be observed, all computational instructions are executed in a single cycle, while load and store operations take either two cycles (for single word ones), or three (for double precision ones). This is coherent with the ARM documentation on Cortex-M4 which states that the CPU has a 32 bit wide bus connecting it to the main memory. As a consequence it is expected that a double 32 word load/store operation takes an extra cycle with respect to its single precision counterpart.

Examining the instruction latency values and the corresponding RAW-free CPI for sequences of the same instruction, we are able to ascertain when functional units are internally pipelined, i.e., when an instruction with a given latency l achieves a CPI lower than l itself. The first case of this fact taking place is the `ldr` instruction, which has an execution latency of two cycles and an asymptotic CPI of 1 for long instruction sequences. This indicates that the load-store unit is indeed internally pipelined. We note that, by contrast, the `str` instruction only achieves a CPI of 2, indicating that the instruction execution is considered complete by the pipeline only when the memory writeback (taking an extra clock cycle) is done. It is worth observing that the asymptotic CPI of 1.5 which is measured whenever computational and `ldr` instructions are interleaved can be caused by a structural hazard on the write port of the register file. Indeed, interleaving `ldrs` and single cycle computational instructions causes a contention on the RF write port when the results of the `ldr` (which had its execution started one cycle earlier)

TABLE 2. Cortex-M4 CPI measurement results. The left side of the table represents the measured CPI values for instruction sequences satisfying the *f_raw*(·) specialization function, while the right side of the table represents the measured CPI values of the sequences chosen to satisfy the *f_noraw*(·) specialization function. The color coding of the values is the following: in green are represented the minimum achievable CPI values, corresponding to 1/*w*, in orange are represented values which reach a CPI greater than the minimum and in red are represented values which achieve more than the double of the minimum achievable CPI.

	$\mathtt{CPI}(f_{raw}(\langle\iota_0\iota_1\rangle))$				$\mathtt{CPI}(f_{noraw}(\langle\iota_0\iota_1\rangle))$				
$\iota_1 \rightarrow$ $\iota_0 \downarrow$	`nop`	`mov/add[i]/addsh` `mul/smlal/lsl[i]`	`ldr` `str`	`ldrd` `strd`	`nop`	`mov/add[i]/addsh` `mul/smlal/lsl[i]`	`ldr`	`str`	`ldrd` `strd`
`nop`	-	-	-	-	1.0	1.0	1.0	1.0	2.0
`mov`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`add[i]`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`addsh`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`mul`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`smlal`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`lsli`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`lsl`	-	1.0	2.0	-	1.0	1.0	1.5	1.5	2.0
`ldr`	-	2.0	2.0	-	1.0	1.5	1.0	1.5	2.5
`str`	-	2.0	2.0	-	1.0	1.5	1.5	2.0	2.5
`ldrd`	-	-	-	-	2.0	2.0	2.5	2.5	3.0
`strd`	-	-	-	-	2.0	2.0	2.5	2.5	3.0

TABLE 3. Cortex-M4 execution latencies for increasing length instruction sequences, composed as a sequence of a single type of instruction.

Instruction	$Lat(f_{raw}(\iota)^n)$ **for** $1 \leq n \leq 9$								
`ldr`	2	4	6	8	10	12	14	16	18
`str`	2	4	6	8	10	12	14	16	18
`ldrd`	3	6	9	12	15	18	21	24	27
`strd`	3	6	9	12	15	18	21	24	27
All others	1	2	3	4	5	6	7	8	9

and of the computational instruction have to be written back.

Given the single issue nature of the Cortex-M4, the number of read ports and write ports is the one of the most demanding operation, namely the double precision Multiply-and-ACcumulate, that requires 4 read ports and 2 write ports to be completed in a single cycle. The inter-stage IS/EX register is thus expected to be 4, 32-bit words wide, while the EX/WB register is not expected to be present, due to the Cortex-M4 3-stages pipeline.

Having completed our microarchitectural inference procedure we can thus draft the alleged Cortex-M4 pipeline structure we have been able to infer. Figure 5 depicts the said pipeline, highlighting in red the inferred components, and in yellow the ones containing synchronous elements about which we can verify the presence of side channel leakage.

We report the results of the side-channel analysis confirmation of the presence of the leakage of the inferred synchronous pipeline elements in Figure 6. In particular, Figure 6 reports the sample Pearson correlation coefficient between the instantaneous power consumption of the device, proportional to its EM emissions, and the Hamming weight of a set of modeled values reported in the right table of Figure 6. The first point to be observed is the confirmation of the leakage related to the RF switching activity, and

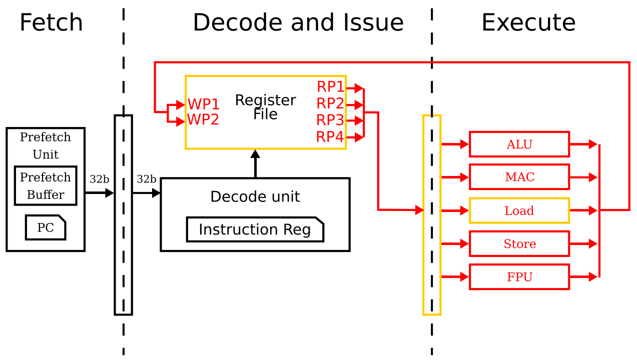
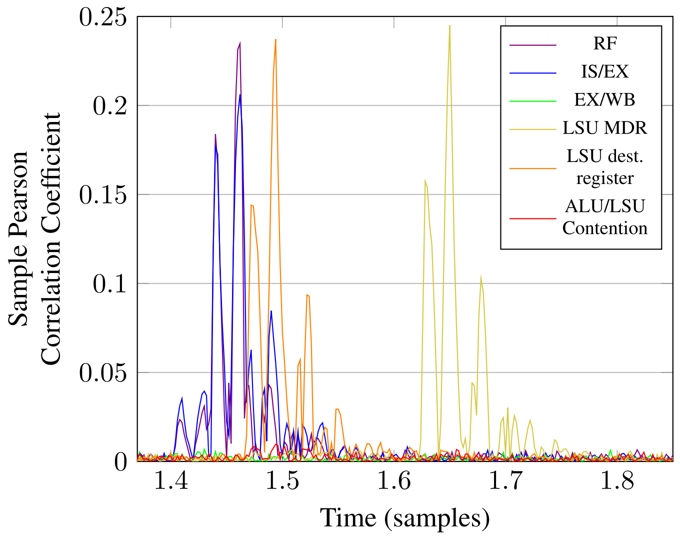


FIGURE 5. Block diagram of the inferred information of the Cortex-M4 processor datapath architecture. In red are highlighted the components that were correctly identified through our feature inference. In yellow are highlighted the components that contain synchronous datapath registers, whose leakage behavior has been confirmed.

TABLE 4. Deduced features of the Cortex-M4 functional units.

ι	$Lat(f_{noraw}(\iota))$	**Pipelined**
`nop`	1	False
`mov`	1	False
`add[i]`	1	False
`addshf`	1	False
`mul`	1	False
`smlal`	1	False
`lsl[i]`	1	False
`ldr`	2	True
`str`	2	False
`ldrd`	3	False
`strd`	3	False

the IS/EX interstage register, which matches the Hamming distance between two subsequent stored values in the said memory elements. Subsequently, we confirm the absence of leakage from the EX/WB register as no side channel leakage compatible with the Hamming distance between two subsequent `add` operation results storing results in different registers is present. This matches the public information on the



Synchronous component	Microbenchmark	Power Model	Verified
RF	`mov r0, r1`	r0 ⊕ r1	✓

IS/EX	`mov r0, r1`
	`mov r2, r3`
		r1 ⊕ r3	✓

EX/WB	`add r0, r1, r2`
	`add r3, r4, r5`
		(r1 + r2)⊕
		(r4 + r5)
			×

LSU MDR	`ldr r0, [x]`
	`3 × (ALU Ops)`
	`ldr r2, [y]`
		[x] ⊕ [y]	✓

LSU dest. register	`ldr r0, [x]`
	`ldr r2, [y]`
		r0 ⊕ r2	✓

ALU/LSU contention	`ldr r0, [r1]`
	`mov r2, r3`
		r1 ⊕ r3	×

FIGURE 6. On the left, it is depicted the result of the synchronous datapath registers verification process, applied on the Cortex-M4. The result shows the value of the Pearson correlation coefficient, computed between the HD of the registers value updates and the captured EM traces. Correlation peaks represent a confirmed leakage behaviour. On the right a summary of the microbenchmarks eliciting the said information leakages.

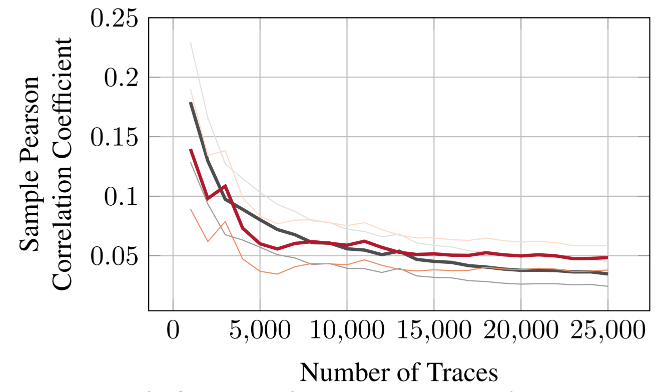


FIGURE 7. Result of a CPA attack against an unprotected AES implementation, we compare the Pearson correlation coefficient of the correct key (colored in thick red), with the second best candidate (colored in grey). The right key byte becomes statistically distinguishable after ∼ 10000 traces.

Cortex-M4 pipeline lacking being a 3 stages pipeline. We then observe the presence of an information leakage attributable to the MDR present in the LSU, employing as a microbenchmark a pair of load operations between which three ALU operations are run. Coherently with the timing delay induced by the three additional, single cycle, operations, the peaks in the value of the Pearson correlation coefficients appears 4 cycles later than the one of computational instructions (3 cycles for the added instructions plus one as the load operation writes the contents of the MDR in the second cycle). The leakage from the LSU destination register, which is contained in the RF, can be observed to take place one cycle after the one caused by computational instructions. Finally, we observe the absence of side channel leakage modeled by the Hamming distance between the target address of the `ldr` operation and a subsequent `mov` operation, suggesting the operand paths are distinct.

To validate the practical impact of the identified leakage, we mounted a CPA attack over an unprotected software AES implementation, executed on the Cortex-M4 core, thus confirming how the EM leakage enables successful key-retrieval attacks. We executed a CPA attack on the first byte of the AES key, by computing 25000 encryptions of random plaintext on the target, capturing for each execution an EM leakage trace, and computing the Pearson correlation coefficient between the traces and the values extracted from a model of the power consumption. The model is built with the Hamming weights of the values computed at the SubBytes stage of the second AES round. The attack can be independently applied on all the subsequent bytes of the key, leading to a full key recovery. The results of the attack are shown in Figure 7.

D. MICROARCHITECTURAL INFERENCE ON CORTEX-M7

We start our inference procedure on the Cortex-M7 confirming the value of the issue width, i.e. $w = 2$, reported in the component datasheet [26]. To this end, we derive all the values of $\texttt{CPI}(f_{noraw}(\langle \iota_0 \iota_1 \rangle))$, and report them in Table 5. In order to avoid issues from caching mechanisms, which would provide a non deterministic access to the main memory, we run our microbenchmark code and place our microbenchmark data in the Tightly Coupled Memory (TCM) provided by Cortex-M7, and implemented in the STM32F746ZG. The TCM is designed to provide deterministic, single cycle access to its contents. As we observe, the minimum achieved CPI value is indeed $\frac{1}{2}$, confirming the dual-issue nature of Cortex-M7.

Before analyzing the results obtained in terms of CPI and latency, it is useful to recall some facts on the TCM as reported by ARM [27]. The TCM is split in two portions, an Instruction TCM (ITCM) and a Data TCM (DTCM). To clearly interpret the results we obtain, it is useful to know that the Data TCM is split onto two banks, D0TCM and D1TCM, and the bank selection is made using the third least significant bit of the accessed address [27]. As a consequence,

TABLE 5. **Cortex-M7 CPI($f_{noraw}(\langle\iota_0\iota_1\rangle)$) measurement results. The color coding is as follows: in green are the minimum achievable CPI values, corresponding to $1/w$, in orange are the values which reach a CPI greater than the minimum and in red are the values which achieve more than the double of the minimum CPI. Load/store instructions marked with a $*$ make use of an address offset equal to 4, while unmarked instructions in which the address offset is set to zero. Base addresses are 32 bit aligned.**

	CPI($f_{noraw}(\langle\iota_0\iota_1\rangle)$)													
$\iota_1 \rightarrow$ $\iota_0 \downarrow$	nop/lsli mov/add[i]	addsh	mul/smlal	lsl	ldr ldrb	ldr* ldrb*	ldrd	ldrd*	str	str*	strb	strb*	strd	strd*
nop/lsli	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	0.5	0.5	1.5	1.5	1.0	1.5
mov/add[i]	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	0.5	0.5	1.5	1.5	1.0	1.5
addsh	0.5	1.0	0.5	0.5	0.5	0.5	1.0	1.0	0.5	0.5	1.5	1.5	1.0	1.5
mul/smlal	0.5	0.5	1.0	0.5	0.5	0.5	1.0	1.0	1.0	1.0	1.5	1.5	1.0	1.5
lsl	0.5	0.5	0.5	0.5	0.5	0.5	1.0	1.0	0.5	0.5	1.5	1.5	1.0	1.5
ldr/ldrb	0.5	0.5	0.5	0.5	1.0	0.5	1.0	1.0	2.5	0.5	3.5	1.5	3.0	1.5
ldr*/ldrb*	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	0.5	2.5	1.5	3.5	3.0	3.0
ldrd	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	3.0	4.0	4.0	3.0	3.0
ldrd*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	1.5	4.0	3.0	3.5
str	0.5	0.5	1.0	0.5	2.5	0.5	3.0	1.0	1.0	1.0	2.0	1.5	1.0	1.5
str*	0.5	0.5	1.0	0.5	0.5	2.5	3.0	3.0	1.0	1.0	1.5	2.0	1.0	1.5
strb	1.5	1.5	1.5	1.5	3.5	1.5	4.0	1.5	2.0	1.5	3.0	1.5	2.0	2.0
strb*	1.5	1.5	1.5	1.5	1.5	3.5	4.0	4.0	1.5	2.0	1.5	3.0	2.0	2.0
strd	1.0	1.0	1.0	1.0	3.0	3.0	3.0	3.0	1.0	1.0	2.0	2.0	1.0	1.5
strd*	1.5	1.5	1.5	1.5	1.5	3.0	3.0	3.5	1.5	1.5	2.0	2.0	1.5	2.0

TABLE 6. **Cortex-M7 execution latencies for increasing instruction sequences lengths, composed as a sequence of a single type of instruction. We do not include store instructions in this table since they have no output register operand, therefore it's impossible to compose an instruction sequence satisfying $f_{raw}(\cdot)$. In the add with shift instructions marked with a †, the RAW conflict is applied on the shifted register, conversely, in the unmarked add with shift, the RAW is placed on the non-shifted register. In load instructions marked with a $\diamond$, the RAW conflict is applied on the offset operands; RAW conflicts are applied to the address operand otherwise.**

Instruction	$Lat(f_{raw}(\iota)^n)$ for $1 \leq n \leq 9$								
mov,add[i]	1	1	2	3	4	5	6	7	8
lsl[i]	1	1	2	3	4	5	6	7	8
addsh	1	2	3	4	5	6	7	8	9
addsh†	1	3	5	7	9	11	13	15	17
mul/smlal	1	3	5	7	9	11	13	15	17
ldr	1	3	5	7	9	11	13	15	17
ldrd	1	3	5	7	9	11	13	15	17
ldr$^\diamond$	1	4	7	10	13	16	19	22	25
ldrb, ldrb$^\diamond$	1	4	7	10	13	16	19	22	25

TABLE 7. **Cortex-M7 CPI($f_{raw}(\langle\iota_0\iota_1\rangle)$) measurement results. The color coding is as follows: in green are the minimum achievable CPI values, corresponding to $1/w$, in orange are the values which reach a CPI greater than the minimum and in red are the values which achieve more than the double of the minimum CPI. In the addsh instructions marked with a †, the RAW conflict is applied on the shifted register, in the unmarked add with shift, the RAW is placed on the non-shifted register. Instructions marked with a $*$ make use of an address offset, conversely to unmarked instructions in which the address offset is set to zero. In load instructions marked with a $\diamond$, the RAW conflict is applied on the offset operands. RAW conflicts are applied to the address operand otherwise. Base addresses are 32 bit aligned.**

	CPI($f_{raw}(\langle\iota_0\iota_1\rangle)$)						
$\iota_1 \rightarrow$ $\iota_0 \downarrow$	mov add lsl[i]	addi	addsh	addsh†	mul smlal	ldr$^\diamond$	ldrb$^\diamond$
mov/add	1.0	1.0	1.0	1.5	1.5	1.5	2.0
lsl[i]	1.0	1.0	1.0	1.5	1.5	1.5	2.0
addi	1.0	1.0	1.0	1.5	1.5	1.5	2.0
addsh	1.0	1.0	1.0	1.5	1.5	2.0	2.0
addsh†	1.5	1.5	1.5	2.0	2.0	2.0	2.5
mul/smlal	1.5	1.5	1.5	2.0	2.0	3.0	3.0
ldr$^{*\diamond}$	1.5	1.5	2.0	2.0	3.0	3.0	3.0
ldrb$^{*\diamond}$	2.0	2.0	2.0	2.5	3.0	3.0	3.0

it is only possible to read two 32-bit words in parallel from the DTCM if they are stored in different banks, as DTCM banks have a single read port, while the CPU is endowed with 2 32 bit read ports towards DTCM [26].

After obtaining the results on the CPI achieved by instruction pairs with (Table 7) and without RAW (Table 5) hazards between them, and the value of the instruction execution latencies $Lat(f_{raw}(\iota)^n)$ for $1 \leq n \leq 9$ (Table 6), we move onto the identification of the number of available functional units, and their latency. We note that Table 7 does not report the same number of cases as Table 5, since it is not possible to cause the RAW predicate to hold for all pairs of instructions in an instruction sequence where store instructions are present. Indeed, since store instructions (str/strb/strd) have no output register, they cannot cause a RAW conflict with the instruction following them. First of all, we observe that the Cortex-M7 is able to reach a CPI of $\frac{1}{2}$ for any combination of computational instructions in our relevant instruction set, save for the case of add instructions with a shifted operand (addsh) and mul/smlal instructions. From this we can infer the presence of a single ALU able to compute addsh (ALU0 from now on) and a single one capable of mul/smlal (ALU1). The second fact is consistent with what is described in the Cortex-M7 datasheet [26], where the presence of a single MAC capable unit is described. We also infer the presence of independent ALUs to be able to sustain a CPI of $\frac{1}{2}$, plus an independent LSU, as all the computational instructions can be dual-issued with a load instruction. We now analyze the latencies of the functional units, and the presence of forwarding paths. Starting from

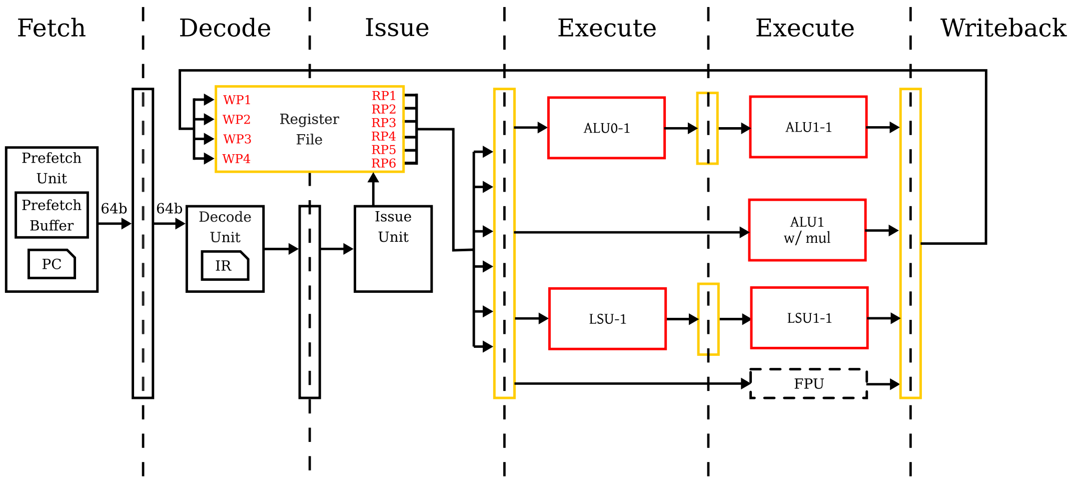


FIGURE 8. Inferred structure of the Cortex-M7 pipeline. In red there are highlighted the datapath components that were correctly inferred in this work, in yellow there are highlighted the synchronous components whose leakage has been experimentally confirmed.

ALU0, we observe that the latency of addsh instructions for a sequence of two or more of them raises by 2 clock cycles per instruction. In contrast with this, the latency of a single addsh is of a single clock cycle. This fact, in conjunction with the capability of the Cortex-M7 to sustain a unitary CPI for addsh whenever no RAW conflicts are present points to ALU0 being a 2 stage pipelined ALU, with its first stage being able of performing a shift operation in a single cycle. This description matches the one of the first ALU described in the described in the Cortex-M7 datasheet [26] (Main/ALU #1 Pipeline in the datasheet). Observing that CPI(f_{raw}(⟨addsh, addsh⟩)) changes between 1 and 1.5 depending on whether the RAW conflict is present among the shifted operands or the non-shifted operands we are able to detect the presence of an EX-EX forwarding path for ALU0. We now analyze the behaviour of the mul/smlal capable ALU1. Considering the unitary CPI in case of a sequence of mul/smlal and the fact that a sequence of mul/smlal operation behaves in the same fashion as one of addsh for what the latency of n operations with RAW conflict concerns, two possible architectures are possible. Either ALU1 matches the structure of ALU0, or it is a single-cycle combinatorial ALU, but no EX-EX forwarding path is present. Considering the description of ALU1 in the Cortex-M7 datasheet [26] (ALU #2 Pipeline in the datasheet) we are able to directly know that ALU1 is a single cycle mul/smlal unit with and thus resolve the point. In the hypothetical case where this point could not have been resolved via public information, we would have proceeded to perform side channel leakage tests for both cases, and confirm the correct one. Finally we analyze in detail the behaviour of the LSU. Starting from the data available in Table 5, we observe that 32 bit load operations (ldr) can be dual-issued with other ldr operations, only when they employ as a target an address belonging to a different DTCM bank (namely, a ldr operation with the third LSB of the address set and a ldr operation with the third LSB of the address clear). By contrast, a sequentialization of the operations, due to a structural hazard in the access to the DTCM causes the CPI to reach 1 in case both loaded addresses belong to the same DTCM bank. An analogous behaviour is reported for the 32 bit store (str) instructions. The load instruction loading a single byte ldrb behave in the same fashion as the ldr instructions and have no structural conflicts with them. The store instruction storing a single byte requires an additional load operation of the word into which the byte should be stored, as the CPU-DTCM bus is 32 bits wide, and does not allow individual byte transfers. This in turn raises the CPI of the strb instructions as a (non parallelizable) hidden load must be performed. Double-word load and store instructions (ldrd/strd) achieve a CPI of 1, which is consistent with the ability of the Cortex-M7 to perform two 32 bit word memory operations toward the DTCM, provided that they reside in different banks. Indeed, loading or storing 64 bits from a 32 bit word-aligned address as in our benchmarks will always result in two memory operations on different DTCM banks.

Finally, we derive the number of read and write ports of the register file, and the interstage register size. Given that the Cortex-M7 is able to issue concurrently a smlal and an add instruction, the total number of 32 bit read operands from the register file is 6, while, to cope with the needs of storing concurrently the outcome of a smlal and a strd (as it happens analyzing a sequence of such operations unaffected by RAW conflicts) 4 write ports are needed. We note that our inference matches the feature description of the Cortex-M7 datasheet [26].

We provide a graphical depiction of the alleged structure of the pipeline of the ARM Cortex-M7, according to our microarchitectural inferences in Figure 8. We note that our deductions are aligned with the publicly available information available in the datasheet, i.e., the Cortex-M7 is a dual issue, in-order CPU, with two ALUs, and an independent

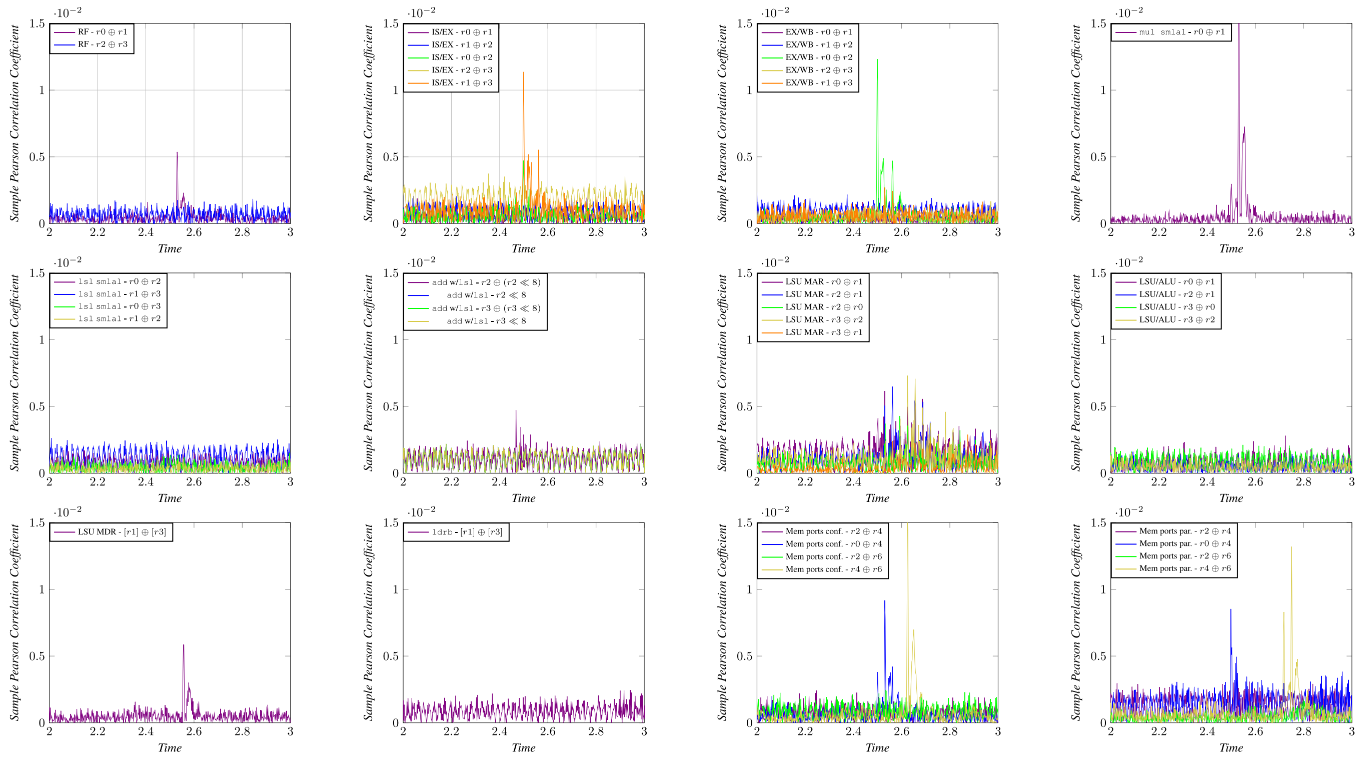


FIGURE 9. Complete report of the detected correlation between the synchronous component switching activity predicted from our inferred microarchitecture, and the benchmark contents. Benchmarks depicted in the same order as listed in Table 8.

LSU and a 6 stages pipeline, of which 2 stages are devoted to EX. We note that all the instructions have $Lat(f_{noraw}(\iota))$ since, in case no RAW conflicts are present, pipelined units allow for an instruction issue per clock cycle.

We now proceed to validate the structure of the pipeline reported in Figure 8 through dedicated microbenchmarks. Table 8 reports a summary of the synchronous component target of the benchmark, the values of which the Hamming weight is employed as the power consumption model and whether or not the model matches actual side channel leakage. Figure 9 reports the quantitative values of the correlation as a function of time; in particular a plot is reported for each synchronous component being benchmarked. Each plot superimposes the correlation of the power consumption models of Table 8. All the correlation values were obtained sampling 1 million traces from the device under examination.

The first three microbenchmarks concern the information leakage coming from the switching activity of the Register File (RF), the IS/EX interstage register and the EX/WB interstage register. The benchmark on the RF has effectively detected correlation between the switching activity of the RF and the power consumption itself, although the correlation with the register move between `r2` and `r3` appears to be quantitatively smaller. The benchmark detecting the information leakage coming from the IS/EX interstage register shows a clear correlation between the (source) operands of the instruction which are being issued in the same pipeline, while no correlation between the (source) operands of different pipelines. This is coherent with the IS/EX register holding them separately. We note a persistent, small but not statistically negligible, information leakage correlated to the distance between `r2` and `r3` is present even when the pipeline is executing only `nops`. An analogous situation takes place when detecting the leakage of the EX/WB interstage register, where the switching activity related to the destination operands, `r0,r2`, of the instructions issued in the first execution lane shows a clear correlation, while the switching activity due to the destination operands of the second execution pipeline, `r1,r3` provides far less side channel leakage (the maximum correlation value is coherent in timing and about three times higher than the threshold for statistical artifacts, $\frac{4}{\sqrt{10^6}} \approx 0.001$). In the `mul smlal` benchmark we test that the multiplier unit is performing both operations is indeed the same, through testing the switching activity due to a single operand change. The corresponding benchmark shows significant correlation. The `lsl smlal` is similar in spirit, willing to test the orthogonality of the two execution pipelines, the one for shifts and the one for multiply-and-accumulate instructions. Indeed, no significant spike in the correlation values is detected in correspondence with the time instant where the operations are performed. The `add` with `lsl` benchmark aims at detecting the leakage stemming from the interstage register present in the 2 stages, pipelined ALU0. Indeed, we have that the device provides statistically significant information leakage related to the distance between the shifted operand, and its original value in an `add`

TABLE 8. Results of the validation of the presence of synchronous components in the Cortex-M7 pipeline via side channel correlation.

Buffer	Stimulus	Power Model	Verif.

RF	`mov r0,r1`
	`mov r2,r3`
		$r0 \oplus r1$
		$r2 \oplus r3$
			✓
			×

IS/EX	`mov r4,r0`
	`mov r5,r1`
	`mov r6,r2`
	`mov r7,r3`
		$r0 \oplus r1$
		$r1 \oplus r2; r0 \oplus r2$
		$r2 \oplus r3; r1 \oplus r3$
			×
			×; ✓
			×; ✓

EX/WB	`add r0,r0,r4`
	`add r1,r4,r1`
	`add r2,r2,r4`
	`add r3,r4,r3`
		$r0 \oplus r1$
		$r1 \oplus r2; r0 \oplus r2$
		$r2 \oplus r3; r1 \oplus r3$
			×
			×; ✓
			×; ✓

`mul smlal`	`mul r0,r0,#1`
	`smlal #0,#0,r1,#1`
		$r0 \oplus r1$	✓

`lsl smlal`	`lsl r4,r0,r1`
	`smlal r5,r6,r2,r3`
		$r0 \oplus r2; r1 \oplus r3$
		$r0 \oplus r3; r1 \oplus r2$
			×; ×
			×; ×

`add w/lsl`	`add r0,r1,r2,lsl#8`
	`add r0,r1,r3,lsl#8`
		$r2 \oplus (r2 \ll \#8)$;
		$(r2 \ll \#8)$
		$r3 \oplus (r2 \ll \#8)$;
		$(r3 \ll \#8)$
			✓; ×
			×; ×

LSU MAR	`ldr r4,[r0]`
	`ldr r5,[r1]`
	`ldr r6,[r2]`
	`ldr r7,[r3]`
		$r0 \oplus r1$
		$r2 \oplus r1; r2 \oplus r0$
		$r3 \oplus r2; r3 \oplus r1$
			✓
			✓; ✓
			✓; ✓

LSU/ALU	`add r4,r8,r0`
	`ldr r5,[r8, r1]`
	`add r6, r2, r0`
	`ldr r7,[r3, r8]`
		$r0 \oplus r1; r2 \oplus r1$
		$r3 \oplus r0; r3 \oplus r2$
			×; ×
			×; ×

LSU MDR	`ldr r0,[r1]`
	3 × (ALU Ops)
	`ldr r2,[r3]`
		$[r1] \oplus [r3]$	✓

`ldrb`	`ldrb r0,[r1, #1]`
	`ldrb r2,[r3, #1]`
		$([r1]$ &
		(0x00FFFFFF)) $\oplus$
		$([r3]$ &
		(0x00FFFFFF))
			×

Mem conf.	`ldr r0,[r1, #0]`
	`str r2,[r3, #0]`
	`ldr r4,[r5, #0]`
	`str r6,[r7, #0]`
		$r2 \oplus r4; r0 \oplus r4$
		$r2 \oplus r6; r4 \oplus r6$
			×; ✓
			×; ✓

Mem par.	`ldr r0,[r1, #0]`
	`str r2,[r3, #4]`
	`ldr r4,[r5, #0]`
	`str r6,[r7, #4]`
		$r2 \oplus r4; r0 \oplus r4$
		$r2 \oplus r6; r4 \oplus r6$
			×; ✓
			×; ✓

operation with left shift of an operand. This in turn confirms the presence of an internal register.

We finally turn to the analysis of the information leakage coming from the load-store unit. The LSU MAR benchmark aims at detecting if the switching activity of the LSU MAR(s) provides information on the side channel. Figure 9 reports that a significant correlation exists between the power consumption and the Hamming weight of addresses employed in consecutive memory operations, while less evident, but still statistically significant correlation exists between addresses of load operations issued either both as the first or both as the second of an issuing bundle. From this result we can infer that there is a logic component whose switching activity depends on the sequence of addresses transiting through the LSU, providing more significant information leakage than the MAR themselves.

Subsequently, the LSU ALU benchmark tests if the adder available in the ALU is also employed to perform address-offset computations. From our microarchitectural inference, we expect it not to be so, and the experimental benchmarks confirm our inferences. In the LSU MDR benchmark we aim at detecting if the MDR of the LSU leaks, per se, information via side channel. To single out its switching activity from the rest of the pipeline we insert three ALU operations acting on orthogonal registers between two `ldr` operations and test the correlation with the Hamming distance of the loaded values. Figure 9 indeed reports correlation with the said Hamming distance in the time instant corresponding to the execution of the second stage of the LSU, i.e. the loading of the word from the memory. Willing to test if the realignment of loaded values is performed by a combinatorial circuit, and thus less likely to provide a punctual leakage in time, we test the correlation with the Hamming distance between two bytes loaded with two `ldrb` operations, acting on addresses containing random 32b values. The corresponding benchmark provides no correlation, suggesting the absence of a synchronous component storing the actual single byte. Finally, we discern if the two MDR, bound to the two data memory interfaces of Cortex-M7 are orthogonal. To this end, we devise a microbenchmark performing a parallel load/store employing the two data memory ports (*memory par.* in Table 8). The benchmark tests for the correlation between all the possible pair of loaded values residing at two addresses belonging to different DTCM banks. As reported in Figure 9, we obtain a significant correlation when employing as a model the Hamming distance between the words stored in the same DTCM bank, in turn pointing to the fact that the four involved loads/stores operations employ two separate MDRs. Finally, we consider the case where a structural hazard on the access to the DTCM banks prevents the execution of two load/store operations at a time (*mem conf.* in Table 8). Two possible outcomes are present: the bank conflict detection is done at issue time, and only a single load/store is issued, or the delay is inserted by the LSU, which accepts the parallel operations, resolves the bank conflict, and delays them. In the former case, we expect to detect correlation between the Hamming weights of the loaded/stored values of adjacent operations, as they are serialized onto the same MDR. In the

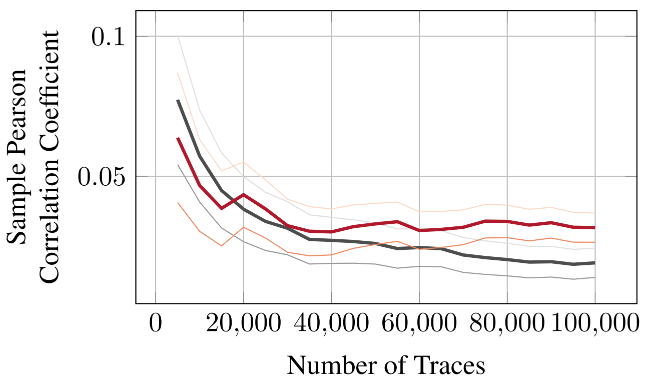


FIGURE 10. Result of a CPA attack against an unprotected AES implementation, running on our Cortex-M7 platform. We compare the Pearson correlation coefficient of the correct key (colored in thick red), with the second best candidate (colored in grey). The right key byte becomes statistically distinguishable after ≈ .75k traces.

latter case, we expect the same leakage pattern as *memory par.*, but with the second relevant leakage taking place later in time. As it can be seen comparing the last two bottom right plots in Figure 9, we can deduce that the bank conflict is detected by the LSU, and two independent MDRs are used also in this case.

Finally, we assess the effectiveness of our leakage model by performing a side-channel attack on an unprotected AES implementation, running on the Cortex-M7. The results of the said attack is depicted in Figure 10. We performed a CPA attack on the first byte of the AES key, by running 100k executions of the cipher on the target, and computing the Pearson correlation coefficient with a model of the power consumption. The model, as in the Cortex-M4 case, is built with the Hamming weights of the values computed at the SubBytes stage of the second AES round. We confirmed how the side-channel leakage enables the retrieval of the first key byte with $\approx$ 75k traces.

V. CONCLUSION

In this work we introduce a complete framework for extracting a side-channel centric microarchitectural leakage model from any unknown in-order CPU. The only prerequisites to our technique are the knowledge of the ISA, the means to obtain the cycle count between two instructions, and a side channel measurement setup to evaluate the hypothesized EM leakage. We validated the proposed methodology on two CPUs of the Cortex-M class, obtaining concrete leakage models, which can be employed in a software only pre-screening for side channel vulnerabilities. Our models highlight the pressing need to consider a microarchitectural view on the side channel leakage, as it highlights potential avenues for attacks, which are not easily caught by examining the ISA-level (i.e., assembly-level) implementation of a cryptographic primitive.

REFERENCES

[1] G. Agosta, A. Barenghi, and G. Pelosi, "Compiler-based techniques to secure cryptographic embedded software against side-channel attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1550–1554, Aug. 2020, doi: 10.1109/TCAD.2019.2912924.

[2] ARM Limited. (2019). *Cortex-M4 Instruction Set Summary*. Accessed: Sep. 9, 2021. [Online]. Available: https://developer.arm.com/docs/dui0553/latest/the-cortex-m4-instruction-set/instruction-set-summary

[3] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers," in *Proc. 19th USENIX Secur. Symp.*, Washington, DC, USA, Aug. 2010, pp. 307–322. [Online]. Available: http://www.usenix.org/events/sec10/tech/full_papers/Backes.pdf

[4] J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F. Standaert, "On the cost of lazy engineering for masked software implementations," IACR Cryptol. ePrint Arch., Tech. Rep. 2014/413, 2014. [Online]. Available: http://eprint.iacr.org/2014/413

[5] A. Barenghi, W. Fornaciari, G. Pelosi, and D. Zoni, "Scramble suit: A profile differentiation countermeasure to prevent template attacks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 9, pp. 1778–1791, Sep. 2020, doi: 10.1109/TCAD.2019.2926389.

[6] A. Barenghi and G. Pelosi, "Side-channel security of superscalar CPUs: Evaluating the impact of micro-architectural features," in *Proc. 55th Annu. Design Automat. Conf.*, San Francisco, CA, USA, Jun. 2018, p. 120, doi: 10.1145/3195970.3196112.

[7] O. Bronchain and F. Standaert, "Side-channel countermeasures' dissection and the limits of closed source security evaluations," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 2, pp. 1–25, 2020, doi: 10.13154/tches.v2020.i2.1-25.

[8] I. Buhan, L. Batina, Y. Yarom, and P. Schaumont, "SoK: Design tools for side-channel-aware implementations," Cryptol. ePrint Arch., Tech. Rep. 2021/497, 2021. Accessed: Sep. 30, 2021. [Online]. Available: https://ia.cr/2021/497

[9] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Proc. Annu. Int. Cryptol. Conf.* in Lecture Notes in Computer Science, vol. 1666, M. J. Wiener, Ed. Santa Barbara, CA, USA: Springer, 1999, pp. 398–412, doi: 10.1007/3-540-48405-1_26.

[10] Z. Chen, A. Sinha, and P. Schaumont, "Using virtual secure circuit to protect embedded software from side-channel attacks," *IEEE Trans. Comput.*, vol. 62, no. 1, pp. 124–136, Jan. 2013, doi: 10.1109/TC.2011.225.

[11] D. Das, M. Nath, B. Chatterjee, S. Ghosh, and S. Sen, "STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, McLean, VA, USA, May 2019, pp. 11–20, doi: 10.1109/HST.2019.8740839.

[12] J. Ferrigno and M. Hlavac, "When AES blinks: Introducing optical side channel," *IET Inf. Secur.*, vol. 2, no. 3, pp. 94–98, Sep. 2008, doi: 10.1049/iet-ifs:20080038.

[13] S. Furber, *ARM System-on-Chip Architecture*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2000.

[14] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, in Lecture Notes in Computer Science, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Paris, France: Springer, 2001, pp. 251–261, doi: 10.1007/3-540-44709-1_21.

[15] S. Gao, B. Marshall, D. Page, and T. H. Pham, "FENL: An ISE to mitigate analogue micro-architectural leakage," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2020, no. 2, pp. 73–98, 2020, doi: 10.13154/tches.v2020.i2.73-98.

[16] A. Hartstein and T. R. Puzak, "The optimum pipeline depth for a microprocessor," in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, Anchorage, AK, USA, Y. N. Patt, D. Grunwald, and K. Skadron, Eds., 2002, pp. 7–13, doi: 10.1109/ISCA.2002.1003557.

[17] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2017.

[18] A. Heuser, O. Rioul, and S. Guilley, "Good is not good enough—Deriving optimal distinguishers from communication theory," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, in Lecture Notes in Computer Science, vol. 8731, L. Batina and M. Robshaw, Eds. Busan, South Korea: Springer, 2014, pp. 55–74, doi: 10.1007/978-3-662-44709-3_4.

[19] P. C. Kocher, "Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems," in *Proc. Annu. Int. Cryptol. Conf.*, in Lecture Notes in Computer Science, vol. 1109, N. Koblitz, Ed. Santa Barbara, CA, USA: Springer, 1996, pp. 104–113, doi: 10.1007/3-540-68697-5_9.

[20] P. C. Kocher, J. Jaffe, and B. Jun., "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf.*, in Lecture Notes in Computer Science, vol. 1666, M. J. Wiener, Ed. Santa Barbara, CA, USA: Springer, 1999, pp. 388–397, doi: 10.1007/3-540-48405-1_25.

[21] D. Kroening, R. Bryant, and O. Strichman, *Decision Procedures: An Algorithmic Point of View* (Texts in Theoretical Computer Science. An EATCS Series). Berlin, Germany: Springer, 2008.

[22] V. Lomné, P. Maurine, L. Torres, M. Robert, R. Soares, and N. Calazans, "Evaluation on FPGA of triple rail logic robustness against DPA and DEMA," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, L. Benini, G. D. Micheli, B. M. Al-Hashimi, and W. Müller, Eds., Nice, France, 2009, pp. 634–639, doi: 10.1109/DATE.2009.5090744.

[23] D. McCann, E. Oswald, and C. Whitnall, "Towards practical tools for side channel aware software engineering: 'Grey box' modelling for instruction leakages," in *Proc. 26th USENIX Secur. Symp.*, E. Kirda and T. Ristenpart, Eds. Vancouver, BC, Canada: USENIX Association, Aug. 2017, pp. 199–216. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mccann

[24] T. S. Messerges, "Power analysis attacks and countermeasures for cryptographic algorithms," Ph.D. dissertation, Univ. Illinois, Chicago, IL, USA, Jan. 2000.

[25] (2020). *ARM Cortex-M4 Datasheet*. Accessed: Sep. 30, 2021. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Processor%20Datasheets/Arm%20Cortex-M4%20Processor%20Datasheet.pdf?revision=904a9fc1-9c66-4816-80bf-ff8c76420e5a&hash=C7B6353BCA7831C236A5509DD62CE8A2

[26] (2021). *ARM Cortex-M7 Datasheet*. Accessed: Sep. 30, 2021. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Processor%20Datasheets/Arm-Cortex-M7-Processor-Datasheet.pdf

[27] (2021). *ARM Cortex-M7 Datasheet*. Accessed: Sep. 30, 2021. [Online]. Available: https://developer.arm.com/documentation/ddi0489/d/Chdeajag

[28] D. Patterson and J. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Amsterdam, The Netherlands: Elsevier, 2011.

[29] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, in Lecture Notes in Computer Science, vol. 6225, S. Mangard and F. Standaert, Eds. Santa Barbara, CA, USA: Springer, 2010, pp. 413–427, doi: 10.1007/978-3-642-15031-9_28.

[30] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn, "IoT Goes nuclear: Creating a Zigbee chain reaction," *IEEE Security Privacy*, vol. 16, no. 1, pp. 54–62, 2018, doi: 10.1109/MSP.2018.1331033.

[31] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic, "EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 71–85, doi: 10.1109/HPCA47549.2020.00016.

[32] H. Seuschek, F. De Santis, and O. M. Guillen, "Side-channel leakage aware instruction scheduling," in *Proc. 4th Workshop Cryptogr. Secur. Comput. Syst.*, M. Brorsson, Z. Lu, G. Agosta, A. Barenghi, and G. Pelosi, Eds., Stockholm, Sweden, Jan. 2017, pp. 7–12, doi: 10.1145/3031836.3031838.

[33] M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, and Y. Yarom, "Rosita: Towards automatic elimination of power-analysis leakage in ciphers," 2019, *arXiv:1912.05183*.

[34] STMicroelectronics. (2018). *Introduction to STM32 Microcontrollers Security*. Accessed: Sep. 30, 2021. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf

[35] (2020). *STSAFE-A100 Authentication & Brand Protection Secure Solution*. Accessed: Sep. 30, 2021. [Online]. Available: https://www.st.com/en/secure-mcus/stsafe-a100.html

[36] F. Terraneo. (2019). *Miosix*. Accessed: Sep. 30, 2021. [Online]. Available: https://miosix.org/

[37] P. Yu and P. Schaumont, "Secure FPGA circuits using controlled placement and routing," in *Proc. 5th IEEE/ACM Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, S. Ha, K. Choi, N. D. Dutt, and J. Teich, Eds., Salzburg, Austria, 2007, pp. 45–50, doi: 10.1145/1289816.1289831.

ALESSANDRO BARENGHI is an Associate Professor with the Politecnico di Milano, Italy. He has published more than 80 papers in international peer-reviewed venues. His research interests include computer and network security, formal languages, and compilers.

LUCA BREVEGLIERI was born in Italy, in 1962. He received the M.Sc. degree in electronic engineering and the Ph.D. degree in electronic engineering of information and systems from the Politecnico di Milano, Milan, Italy, in 1986 and 1991, respectively. From 1991 to 1998, he worked as the Information and Communications Technology (ICT) Manager and a Network Administrator. In 1998, he was appointed as an Associate Professor at the School of ICT, Politecnico di Milano. He researched the design of dedicated architectures for computer arithmetic and signal and image processing. Since he was appointed as an Associate Professor, he has been researching mainly the fields of security of digital devices, both dedicated and programmable, of efficient applied cryptographic algorithms, and of the protection against attacks (mathematical and side channel); and secondarily the field of theoretical computer science (formal languages and automata). He has participated in several European Union (EU) and national research projects (MEDEA+, ENIAC, and PRIN), mostly on the security of digital devices. He collaborated with CERN, Geneva, on the design of a digital to analog converter (DAC) for the particle detectors for large hadron collider (LHC), and with STMicroelectronics and Micron on applied cryptography topics. He is a coauthor of over 100 international peer-reviewed journals and conference papers on his research topics. He supervises the supercomputing activities by his university.

NICCOLÒ IZZO is currently pursuing the Ph.D. degree in information technology with the Politecnico di Milano, Italy. He collaborated with Micron on the application of cryptography on emerging memory devices and co-authoring one patent. His research interests include security in traditional and emerging memory subsystems, and microarchitectural side channel attacks.

GERARDO PELOSI (Member, IEEE) is an Associate Professor with the Politecnico di Milano, Italy. He has published more than 90 papers in international peer-reviewed journals and conference proceedings and is a co-inventor of ten patents concerning the design of cryptographic systems. His main research interests include in computer security, cryptography, security in hardware, and in the area of data security and privacy.

Mobile Systems Secure State Management

Paolo Amato
Micron Technology Inc.
Vimercate, Italy
pamato@micron.com

Niccolò Izzo
DEIB
Politecnico di Milano
Milano, Italy
niccolo.izzo@polimi.it

Carlo Meijer
Institute for Computing and Information Sciences
Radboud University
Nijmengen, The Netherlands
cmeijer@cs.ru.nl

***Abstract*—Today's mobile devices are equipped with sophisticated *chain-of-trust* mechanisms, able to successfully mitigate tampering of critical software components. However, this technology, on the one hand, hinders the permanence of malware, thus raising the complexity for developing rootkits. On the other hand, the freedom of the end-user is limited. In fact, with all the security features enabled, one could not run any privileged code without it being signed by the Original Equipment Manufacturer; modifying any component of the root partition would cause a device read error and small modifications could be even rolled back automatically. Original Equipment Manufacturers typically provide mechanisms to (partially) disable these security features. However, they usually require two conditions: every unlock request must be approved by them, e.g. for warranty implications; secondly, to preserve the device security level, each time a security feature is disabled, the user data must be completely erased. We analyze several bootloader related vulnerabilities which allow to bypass these two requirements by exploiting design and implementation flaws in smartphones from different vendors. We then propose a novel architecture for secure device status storage and management. Our proposal relies only on commodity hardware features, which can be found on most mobile platforms. Furthermore, differently from many commercial implementations, we do not consider the storage device firmware as trusted, this makes our attack surface smaller than all of the examined alternatives.**

***Index Terms*—mobile devices, systems security, security requirements, hardware security protocols**

I. Introduction

Mobile devices like smartphones, tablets and wearable devices are ubiquitous. They generate, process and gather large quantities of personal and sensitive data, as they acquired an instrumental role in daily activities of many, from videos shooting and photo taking to social interaction, to being a gateway for banking and payment services, or even participate in authentication schemes, either standalone or as a second factor. The consequences of a successful breach of a mobile device may include monetary theft, but also intellectual property or identity theft. This makes mobile devices an extremely attractive target for attackers.

Android-based mobile systems are complex devices, where both hardware and software are produced by several parties, with diverse privilege levels. On the least privileged side we have apps built by third-party developers, which are executed within a sandboxed environment. These apps may try to exploit software vulnerabilities to gain root capabilities, and modify the system to keep those permissions as long as possible, even across reboots.

Protection against system modification can be carried out from the file system level (dm-verity [1]) up to a hardware-based root-of-trust, through a mechanism called *chain-of-trust*. It works by having each software component authenticate the next through cryptographic signature verification, and only transferring control to it upon successful verification. The first element of the *chain-of-trust* is stored inside a hardware-enforced read-only memory called the *Boot ROM*. Thus, it cannot in any way be modified. From Android version N (7.0) onward, even a limited form of Forard Error Correction (FEC) [2] is incorporated, to be able to restore a device state after incidental or malicious storage corruption.

However, such mechanisms heavily limit the freedom of the user, who may want to execute custom-built kernels or to voluntarily modify some software or hardware components of his own mobile system. In order to cater to enthusiast users, typically some means of bootloader unlocking is offered, allowing the end-user to disable some or all the security features of the device, after approval by the Original Equipment Manufacturer (OEM). This OEM approval should be a critical step in the unlocking process. On top of that, each time the security level is lowered, the content in user partitions should mandatorily become unreadable, in order to prevent sensitive data from leaking from a lost or stolen device.

Contributions. In this work, we perform a security review of bootloader unlocking mechanisms for several devices from multiple manufacturers. Abstracting from the design flaws exposed by multiple discovered vulnerabilities, we propose a bootloader unlocking architecture, built upon widespread hardware primitives, able to guarantee strong security properties. We also provide a preliminary investigation on a formal proof of its security properties under the Dolev-Yao [3] attacker model.

II. Background

To be able to perform a meaningful security analysis, we start by delimiting our threat model. To better identify the threat model to which smartphones are subject, we split it in two according to the attackers ability to have physical access to the device. The aim of both models is to break the device's chain-of-trust; to do so either the bootloader is turned into an unlocked state or the device itself is transitioned into an engineering sample, which in most of the cases has some critical, or even all the security measures of their production counterparts, disabled.

1) Software-only Attack Model: In this scenario we assume the attacker to be already able to run arbitrary user-land code at the highest privilege level, i.e. as *root* user. Therefore, the motivation for a malicious entity for breaking the device's chain-of-trust might seem limited, as sensitive user data stored on the device is already compromised. However, breaking the chain-of-trust allows to implant covert persistent malware, install custom operating systems, or exfiltrate the secret encryption keys used for Digital Rights Management (DRM).

2) Hardware Attack Model: Smartphones are designed to be portable and small-sized; these characteristics make them a frequent subject of loss and theft. Thus, we are compelled to include a hardware attack model in our analysis. An attacker can gain physical access to a device and either extract the secrets, or even install a rootkit onto the device and covertly return it to the owner, so that further data can be collected in the future, scenario that is well-known under the name of Advanced Persistent Threat (APT). In this model, we assume the attacker has the ability to disassemble the device and is able to rework soldered components, such as the storage medium. Furthermore the attacker may, for example, apply an interposer to the storage module to perform wiretapping and tampering of the host-storage communication, or to read and write the content of the storage component itself. Finally, the attacker may potentially swap the storage medium between the device under attack, and one that is under the attacker's control.

A. Android Security Architecture

The storage medium is a particularly relevant peripheral as it contains all the user data, the operating system files and other sensitive information such as DRM keys, biometric data and device status flags. The storage medium includes a microcontroller, which processes commands received from the System on Chip (SoC) and implements several features such as block device abstraction, access control, bad block management and wear leveling. The communication with the SoC is based on either the embedded Multi Media Card (eMMC) [4] protocol or its successor; the Universal Flash Storage (UFS) [5] protocol.

Additionally, the SoC itself typically embeds the public signing key of the OEM to serve as an anchor for *chain-of-trust* mechanisms, as well as a set of electronic Fuses (eFuses). eFuses can be *burned* only once and their status can be readout, and are used by the OEM to implement software downgrade protection, and for the provisioning of boolean flags. However, many OEMs are reluctant to use eFuses due to their irreversible nature: they hamper the ability to refurbish used devices. Rather, OEMs rely on the storage medium so that the device is restored to a factory-reset state, simply by replacing the storage medium altogether.

To accomodate this need, eMMC and UFS drives can limit temporarily (until the next boot) or permanently, write access to any region. Furthermore, the Replay-Protected Memory Block (RPMB) offers additional security features. It is a small region (4 x 4MB), to which all read and write requests are denied unless accompanied with a valid symmetric signature based on HMAC-SHA256. The secret key is shared between the SoC and the storage medium during the manufacturing process. Furthermore, a *write counter* prevents the contents to be replayed and downgraded to a previous version, as its value is included during the signature generation. RPMB is typically used to store the device's bootloader unlock status, and cryptographic keys such as those used for DRM.

1) Chain-of-Trust: All the popular SoC platforms feature a chain-of-trust validation at boot, to exemplify the process, we describe the one adopted by Qualcomm SoCs [6]. The boot process starts from the execution of a Primary BootLoader (PBL), which is stored in the boot ROM (e.g. part of the SoC silicon mask). The PBL initializes essential hardware such as the eMMC/UEFI storage from which the next boot components can be loaded. The PBL loads the Secondary BootLoader (SBL), a.k.a. the eXtensible BootLoader (XBL) from the storage, verifies its digital signature, and hands over the execution [6]. The XBL initializes more subsystems such as secondary CPU cores and DRAM, reads and extracts the Android BootLoader (ABL) which is a Unified Extensible Firmware Interface (UEFI) compatible Linux bootloader. ABL initializes a large number of Driver eXecution Environment (DXE) modules to perform sophisticated tasks such as updating the eMMC/UFS firmware, initializing the RPMB and displaying graphical elements on the screen. Finally, ABL launches the Linux kernel for booting the Android OS or enters the recovery environment.

Verified Boot is a software framework to extend the chain-of-trust to the entire operating system. It works by verifying all the system partitions. It is usually configured so that only authorized parties (typically OEMs) possess the cryptographic keys required for creating updates to the operating system. Google provides a reference implementation called Android Verified Boot (AVB) [7], which also encompasses an anti-rollback protection mechanism.

Enthusiast users may wish to install an aftermarket operating system on their device, such as LineageOS [8] or PostmarketOS [9]. For obvious reasons, these operating systems are not cryptographically signed by the device OEM, and thus cannot be installed by default. In order to address the demand for aftermarket operating systems, many OEMs offer mechanisms to deliberately break the chain-of-trust.

2) Filesystem Confidentiality and Integrity: Dm-verity is a Linux module which builds a Merkle tree over the root and system partitions at 4KB block granularity, to guarantee integrity on OS partitions. Every time a block is read, its digest is verified up to the root of the tree. Ultimately, the root of the tree is then signed with a cryptographic signature. Dm-verity effectively makes the system partition a part of the chain-of-trust, without the necessity to verify a cryptographic signature over several gigabytes of data each time during the device's boot sequence.

Disk Encryption is a mechanism which guarantees integrity and confidentiality for stored data, in a transparent way. Android supports two types of encryption: Full-Disk Encryption (FDE) and File-Based Encryption (FBE). The former acts at block-device level and requires the user to enter the encryption key at boot; while the latter is performed at filesystem level and can differentiate between *device encrypted storage* and *credential encrypted storage*. The first one is encrypted with a device-specific key and allows the device to boot, the second is encrypted with the user's credentials and is available only when the phone is unlocked, Both features are implemented as kernel modules, so they rely on the kernel being pristine.

3) ARM TrustZone™: TrustZone is a security extension to the ARM instruction set. It is marketed as a low-cost alternative to adding a dedicated security co-processor to the SoC. It works by providing a hardware-backed separation between two processor modes, referred to as the *secure world* and the *normal world*. The secure world typically offers functionality through an API to the normal world, similar to syscalls on general purpose platforms. However, the secure world is not another higher-privileged level built on top of the others. Rather, it is orthogonal to all the other processor capabilities, thus within the secure world, privilege and memory separation can also be taken advantage of. Typically, the regular operating system runs within the normal world, and a much smaller operating system with limited functionality runs within the trusted world, with the ultimate goal of reducing the attack surface. Features implemented by the trusted code often include DRM functionality, rate limiting for device unlocking attempts, biometric unlocking, SIM locks, and more.

III. Related Work

Sean Beaupre's SamDunk [10] technical paper describes a faulty state storage architecture, which relies on the invariance of an identifier derived from the eMMC. That identifier, which is writable-once by standard, was found to be re-writable due to a hidden vendor command, confirming the threats of proprietary closed-source storage firmwares. The vendor commands were first discovered by Oran Avraham [11].

Redini et al. in their Bootstomp [12] work analyze the security of bootloaders from four different vendors by means of symbolic execution, they isolate six new vulnerabilities and derive guidelines for the design of secure bootloaders.

Roee Hay from Aleph Security [13] describes a number of notable bootloaders vulnerabilities also related to OEM customization of existing mobile platforms, they also make several suggestions to reduce the overall attack surface of the fastboot interface.

Although several papers describe bootloader-related vulnerabilities, to the best of our knowledge, no prior work exists that attempts to infer the underlying patters and defines a secure state storage architecture as an alternative.

IV. Vulnerabilities

In this section we offer an overview of security weaknesses found in mobile devices from several manufacturers [1] , drawing some general traits from them which we can keep into account to design a secure alternative.

A. HTC

HTC offers users to unlock their phones by requesting an *unlock token* from **htcdev.com**. Their phones also contain a *security flag* with two states: *S-ON* for consumer phones or *S-OFF* for engineering samples. S-OFF disables cryptographic signature checks in the boot loader, baseband firmware, and TrustZone. For most HTC models, the security flag is stored within the storage medium, with temporary write protection set at each boot, an attacker in a physical access scenario can thus change the flag by de-soldering and accessing the storage medium. If we assume a scenario in which all sensitive user data is encrypted, the key is cryptographically bound to the device's unlock code or pattern, and unlock patterns with sufficient entropy are impractical. Thus, a TrustZone applet performs rate limiting to inhibit brute-force attempts. However, since the device no longer verifies signatures over the TrustZone image, an offline brute force attack becomes feasible. Furthermore, HTC extends *Qualcomm Secure Execution Environment (QSEE)* with several vendor-specific commands. HTC uses a *command bitmask* determining which commands are enabled. At power-on, all commands are enabled, while during boot, a subset of them are disabled [14].

Among the public exploits available to unlock HTC phones are:

a) GFree: [15] created by *scotty2*, communicates with the *Power Management Unit (PMU)*, to power-cycle the eMMC chip, thus, lifting the temporary write-protection, making the security flag writable.

b) RevOne: [16] works by issuing a TrustZone call to `tzbsp_oem_rand`, an HTC-specific command that writes random data to an address parameter supplied by the non-secure world. However, the address is not properly sanitized. RevOne exploits this by calling the function on the command bitmask. Repeated calls are issued until the command `tzbsp_oem_emmc_write_prot` becomes enabled in the bitmask. Then, `tzbsp_oem_emmc_write_prot` is called causing the phone to not apply write-protection over the security flag. Finally, the phone is rebooted and the security flag is set to S-OFF simply by writing to it.

c) Firewater: [17] is developed by *beaups* and *fuses*. The GFree exploit was patched by applying write protection at power on, rather than at boot. However, a small window of opportunity exists between the chip being powered on and its sensitive regions being write-locked. Firewater abuses this by attempting to change the security flag during that time.

[1]All of the described vulnerabilities have been patched either in future software releases of the same product, or in successive products. We consider this to be a proof of the OEMs knowledge of the respective vulnerabilities. Therefore since OEMs are aware of each of the found vulnerabilities, no disclosure process was needed.

HTC has addressed this issue by restricting access to the PMU I/O registers from the non-secure world within TrustZone. Furthermore, the kernel APIs used by Firewater were removed, and finally, some eMMC chips have been provided with firmware updates [17].

d) Sunshine: [18] is a paid solution supporting both HTC and Motorola phones. It contains exploit code for several vulnerabilities in order to support a multitude of devices. Its code is heavily obfuscated in an attempt to prevent researchers and device manufacturers from studying it. As such, many devices running the latest firmware are supported, and at times support is added for new devices [18]. Among the exploits used by Sunshine, we analyze and propose some notable examples.

In case Sunshine is run on an HTC One (m7), and possibly other devices, a vulnerability in another HTC-specific TrustZone function is exploited: `tzbsp_oem_hash`. This function computes a cryptographic hash over its input buffer and copies the output to the address pointed to by one of its input parameters, without being properly sanitized. Similar to RevOne, Sunshine exploits this by pointing the output to the command bitmask. The input buffer is chosen such that the first byte in the hash output equals `0xff`. Every diagnostic command is subsequently enabled by issuing repeated calls, where the output address is incremented by one byte each time. Finally, the security flag is set to S-OFF by means similar to those of RevOne. We found that this method is confirmed to work on the latest firmware. Inspection of firmware images for the One m8 reveals that it too suffered from this vulnerability. However, the issue was addressed in the latest firmware.

B. Motorola

Similar to HTC, Motorola phones can also be officially unlocked by requesting unlock tokens from **motorola.com**. Here the bootloader lock status is represented by an eFuse: intact for locked, blown for unlocked. Thus, unlocking a Motorola phone is irreversible. Motorola extends QSEE with several commands. Including those that allow one to get/set the unlock status. Unlocking the bootloader will only succeed when accompanied with a valid unlock token.

Due to the way Motorola designed their locking mechanism, the situation is less critical. The mandatory factory reset can be circumvented by making a backup of the user data and restoring it after the event. However, this only enables unencrypted data. Since cryptographic signatures over the TrustZone partition are still enforced, a brute-force attack is infeasible due to the rate limiting applied by the TrustZone applet. The ability to unlock Motorola's bootloader by unofficial means effectively depends the discovery of unsigned code execution either within Qualcomm Secure Execution Environment (QSEE), or a Motorola-specific extension thereof [19]–[21].

C. Google

Google, on their latest products, the Pixel 6 and Pixel 6 Pro, allows customers to unlock the bootloader. The official bootloader unlock procedure enforces a wipe of all the user data on the device and does not require OEM approval. From the Pixel 2 onward, Google devices also support user-provided public keys while enforcing verified boot [22], tech-savy users can thus use their keys for secure boot. Google stores the bootloader status and user-provided keys in both the RPMB partition of the storage medium, and, on devices where it is available, in their security co-processor referred to as *Titan M*. Titan M is a discrete device, with a custom silicon design conceived by Google itself which serves as root-of-trust, with a minimized attack surface. While the option of storing the state only on RPMB is flawed due to the necessity of trusting the storage controller firmware, deploying a separate co-processor is a far more secure option. However this option although slowly becoming more community-driven [23], is presently not offered to third-party OEMs. For Google phones equipped with the Titan M secure element, the backup-wipe-restore attack is ineffective, as the user data is always encrypted and the key is erased by the secure element. It is infeasible for an attacker to restore the key from a backup, since it is kept within secure element itself, rather than the storage medium.

D. Samsung eMMC backdoor

Many Samsung eMMC chips contain a vendor-specific command that allows one to perform diagnostic and debugging tasks. Among other functionality, it allows for raw read/write access into the eMMC controller's address space. As such, any desired behavior of the storage chip can be triggered from the host side. This functionality can be leveraged in order to repair phones suffering from a so-called *sudden death syndrome* [11]. Sunshine also uses this functionality. In case a Samsung eMMC chip is detected, it attempts to unlock the write-protected area through a vendor-specific command, and subsequently modifies the security flag.

E. Common Weaknesses

In the Android ecosystem every vendor creates its own custom implementation for managing locked/unlocked and production/development states. As described by the SamDunk write-up document [10], some use TrustZone protected fuse bits, others use write-protected flags stored in the NAND, signed blobs in the stored in the boot partition, etc. This large variety increses the effort for an independent security review of all the implementations. OEM code also frequently lacks sufficient quality assurance and testing necessary in security-critical environments such as TrustZone. This could be solved either by adopting a multi-party model such as CHERI [24] or RISC-V Multi-Zone [25] which would allow one to enforce the least-privilege principle so that entities receive only the privileges they absolutely need. Furthermore, storing the device state flag inside the RPMB partition would have guaranteed integrity even against hardware attackers in HTC devices. RPMB is nowadays supported by most of the eMMC and all of the UFS chips. Using the RPMB partition alone implies trusting the controller firmware, but having it as an additional measure can increase the security level.

V. Secure State Storage

We propose a novel extended bootloader architecture, in which we attempt to address these issues, while providing sufficient flexibility for the OEM in order to promote adoption.

A. Properties

Listed below are the properties guaranteed by the proposed security architecture:

a) Cryptographic unlock procedure: The chain-of-trust can be broken by the OEM if desired. However, this requires a valid cryptographic signature to be issued by the OEM.

b) Multiple states management: Multiple security levels can be defined. For example, the OEM may define an *engineering mode* that, besides the kernel and system partition, also allows for the baseband firmware to be freely modified.

c) Easy and secure refurbishment: The architecture does not rely on a secret shared between the SoC and the storage medium. As such, it is relatively straightforward to replace the storage medium.

d) No trust in storage medium: Without trusting the storage medium, the security architecture survives a full storage controller compromission, such as [10].

e) Security downgrade implies full wipe: Security can not be downgraded unless the user data is erased.

B. Hardware Prerequisites

A mobile platform, to be compatible with the proposed security architecture, must support a set of hardware primitives, whose, to the best of our knowledge, are available on all current mainstream mobile platforms.

a) Trusted Environment: Firstly, an isolated environment is required in which trusted applets that handle secret key material can be executed. This can be implemented in a dedicated co-processor, or a trusted execution environment such as ARM TrustZone. TrustZone has the advantage that it is already integrated in every Cortex-A based device presently on the market. This makes an implementation of the architecture possible also on commercially available devices.

b) One-time-programmable memory: The second requirement is One-Time-Programmable (OTP) memory. As of today, this feature is already ubiquitous in most mobile devices. For example, OTP is used for storing the OEM's public key that serves as anchor for the CoT. The hardware should be able to restrict access from the untrusted environment (i.e., non-TrustZone) to OTP memory. In some hardware architectures, this is the default. In others, either its memory-mapped location, or its I/O addresses, whichever applies, should be restricted. Qualcomm's *External Protection Unit (XPU)*, for example, allows for this. Consequently, the untrusted environment is unable to access the OTP directly. Instead, it is forced to interact with the trusted environment. In case TrustZone is used for isolation, access to OTP memory should be allowed from the highest privilege level only, i.e. the TrustZone kernel. As such, a vulnerability within a TrustZone applet is not enough to access the OTP contents.

c) Replay counter: Lastly, a hardware primitive is needed that allows for invalidating previous states. This mitigates potential replay attacks where previously valid key and signature material is replayed in order to recover a previous version of the Disk Encryption Key (see next section). The primitive is modeled with a counter value that can be incremented, but not decremented. This can be implemented e.g. with a set of eFuses, where an eFuse is burnt each time the counter is incremented.

C. Architecture

We now present the notation that was adopted in the description of the security architecture.

The *Device key (DK)* is a random per-device secret key. Preferably, the DK is generated on the device itself during the provisioning process. It is stored within OTP, never to be changed again. Furthermore, the DK is confined to the TrustZone kernel. Under no circumstance should the DK be recoverable outside of the TrustZone kernel.

The *Shared key (SK)* is another random secret key. It is shared between the device and the OEM. Similar to the DK, it is also confined to the TrustZone kernel.

The *Status IDentifier (SID)*, is a value determining the security state. `LOCKED` represents a fully locked device, with all its security features active. The OEM is free to introduce additional states. For example, `UNLOCKED` for unlocked bootloader, and `ENGINEERING` for engineering sample.

Counter c is the replay counter as described in section V-B.

The *Disk Encryption Key (DEK)* is the symmetric key used for encrypting the user data.

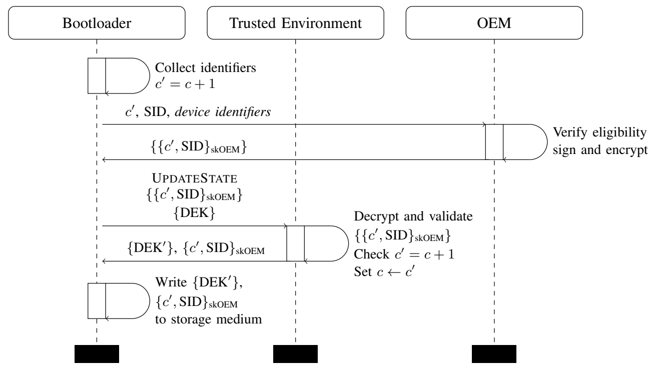
The *Key Encryption Key (KEK)* is used to encrypt the DEK. An example representation is $\mathcal{H}((c, \texttt{LOCKED}), \mathrm{DK})$: a concatenation of counter value c and the SID corresponding to a locked device, fed to keyed hash function $\mathcal{H}$ using DK as the key.

$\{c, \texttt{UNLOCKED}\}_{\mathrm{skOEM}}$ represents the certificate containing counter value c and SID `UNLOCKED`, signed by skOEM, the OEM's asymmetric secret key.

Lastly, $\{x\}$ represents the symmetric encryption of x. The encryption key is clear from the context.

The two main components of the architecture are described below: the unlocking procedure and the process determining the locking status on boot.

1) Unlocking the device: Device manufacturers often prefer to keep track and/or exercise control over which devices have their security state downgraded (e.g. for warranty purposes). This can be achieved in many ways. We choose to encode the device's security state in an asymmetrically signed certificate that is verified each time during the boot process. This introduces negligible overhead, as the Verified Boot process already performs such a verification for each stage of the boot process. Signature checking on every boot has the added benefit of preventing an attacker from permanently downgrading security. Indeed, every hypothetical security breach is temporary until the device is rebooted.



Bootloader
Trusted Environment
OEM
Collect identifiers
$c' = c + 1$
c', SID, *device identifiers*
Verify eligibility
sign and encrypt
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
UPDATESTATE
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
{DEK}
Decrypt and validate
$\{\{c', \mathrm{SID}\}_{\mathrm{skOEM}}\}$
Check $c' = c + 1$
Set $c \leftarrow c'$
{DEK'}, $\{c', \mathrm{SID}\}_{\mathrm{skOEM}}$
Write {DEK'},
$\{c', \mathrm{SID}\}_{\mathrm{skOEM}}$
to storage medium

Fig. 1. Sequence diagram of the proposed unlock process. The request incorporates an OEM-specific set of device identifiers, the desired SID, and counter value $c + 1$. In case the OEM approves the unlock, the request is signed, encrypted and sent back. The response is then passed to the trusted environment for decryption, ensuring c is incremented.

A sequence diagram describing the unlock procedure is depicted in Fig. 1. The unlock request containing all the information required by the OEM is transferred to a server hosted by the OEM. The request incorporates a set of device identifiers chosen by the OEM, the desired SID, and counter value c. After performing some OEM-specific checks to assess the status change eligibility, a signed certificate $\{\{c, \texttt{UNLOCKED}\}_{\mathrm{skOEM}}\}$ is eventually generated, encrypted using the shared key SK, and returned to the user. The response is fed to UPDATESTATE (see section V-C2), which increments replay counter c, and returns the plaintext certificate. Finally, the certificate is stored within the (untrusted) storage medium.

2) State Management: The architecture defines two procedures within the trusted environment: QUERYSTATEANDDEK and UPDATESTATE. They are used during each device's startup procedure, and every time the security state is changed, respectively.

QueryStateAndDEK(Certificate,{DEK})

```
**if** $Verify(Certificate) \wedge$
    $Certificate.c = c \ \wedge \ \ldots$ // OEM-specific checks
        $SID \leftarrow Certificate.SID$
**else**
        $SID \leftarrow \texttt{LOCKED}$
$KEK \leftarrow \mathcal{H}((c, \mathrm{SID}), \mathrm{DK})$
$DEK \leftarrow Decrypt(\{DEK\}, KEK)$
**return** $(DEK, SID)$
```

QUERYSTATEANDDEK returns both the current security state and the decrypted DEK. The certificate and encrypted {DEK} are passed as parameters. Alternatively, they can be retrieved from the storage medium from within the procedure itself. However, in the assumed attacker model (section II-2), the storage medium is not trusted. By passing these values as parameters, we make the untrusted nature more explicit. This has the additional benefit of not requiring storage-related code to be present within the trusted environment (for this purpose at least). Thus, reducing the attack surface.

A trustworthy SID is obtained if and only if both the certificate's signature is valid, and its counter value equals replay counter c. In any other case, the default SID, corresponding to the activation of all security features is selected. Then, the KEK is obtained by taking the concatenation of counter c and SID, and feeding it to keyed hash function $\mathcal{H}$, using DK as the key. The KEK is then used to decrypt the encrypted {DEK}. Finally, both the DEK and SID are returned.

UpdateState(CurrentCert,{NewCert},{DEK})

```
**if** $Verify(CurrentCert) \wedge$
    $CurrentCert.c = c \ \wedge \ \ldots$ // OEM-specific checks
        $CurrentSID \leftarrow CurrentCert.SID$
**else**
        $CurrentSID \leftarrow \texttt{LOCKED}$
$NewCert \leftarrow Decrypt(\{NewCert\}, SK)$
**if** $Verify(NewCert) \wedge$
    $NewCert.c = c + 1 \ \wedge \ \ldots$ // OEM-specific checks
        $NewSID \leftarrow NewCert.SID$
**else**
        $NewSID \leftarrow \texttt{LOCKED}$
**if** $CurrentSID < NewSID$ **then**
        $DEK \leftarrow Random()$
**else**
        $KEK \leftarrow \mathcal{H}((c, \mathrm{CurrentSID}), \mathrm{DK})$
        $DEK \leftarrow Decrypt(\{DEK\}, KEK)$
$KEK' \leftarrow \mathcal{H}((c + 1, \mathrm{NewSID}), \mathrm{DK})$
$\{DEK\}' \leftarrow Encrypt(DEK, KEK')$
$c \leftarrow c + 1$
**return** $(\{DEK\}', NewCert)$
```

The UPDATESTATE procedure is issued after a certificate has been successfully obtained from the OEM. The current DEK is retained or a new one is generated, in case of an upgrade or downgrade of the security state, respectively. Invariably, the counter value c is incremented. Thus, under no circumstance shall a previous version of the DEK be recoverable by feeding a previous version of the {DEK} parameter to the QUERYSTATEANDDEK method.

Note that the device can be reverted to a fully locked state by providing an invalid certificate. Thus, this transaction does not strictly require an involvement of the OEM.

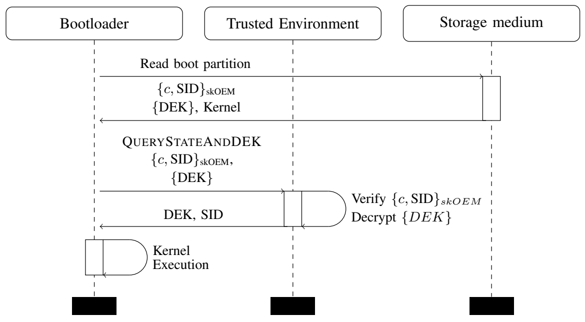


Fig. 2. Sequence diagram of the proposed boot process. The resulting SID is trustworthy and is used to toggle security features, such as Verified Boot and dm-verity.

3) Device Operation: We consider an example implementation of our architecture into an Android device running the latest version at the time of writing, i.e. Android 13 "Tiramisu". The device is equipped with a File-Based Encryption implementation, with a UFS-based storage subsystem. The described boot process is summarized with a sequence diagram in Fig. 2. The boot process starts by retrieving the boot partition from the storage medium. The partition contains, at a minimum, the encrypted $\{\text{DEK}\}$, the kernel to be launched, and signed certificate $\{c, \text{SID}\}_{skOEM}$. The bootloader subsequently invokes the QUERYSTATEANDDEK method providing the certificate. The method determines the effective SID, decrypts the DEK and returns both. The effective SID is then used to determine which security features should be enabled, such as Verified Boot. Once the kernel's integrity has been verified through cryptographic signatures (or such check has been omitted, depending on the effective SID), it is launched and receives the DEK as a parameter. In case the DEK has been decrypted correctly, the Android OS will be able to correctly decrypt the user partition. Nevertheless if the scheme is tampered with, e.g. by manipulating the certificate, the security status is restored to its default state, and if it differs from the intended one, the resulting DEK will be invalid. Thus, the user data remains confidential.

4) Device End-of-Life: Note that the security state identifier SID is not stored within the SoC. Rather, it is provided by the storage medium, but is authenticated nonetheless. As such, erasure or replacement of the storage medium causes the device to reset to its default security state. Hence, the storage medium needs not to be trusted. Thus, questionable functionality for transferring secret keys, such as the RPMB authentication key, from the SoC into the storage medium, becomes unnecessary.

D. Design decisions

The properties listed at the beginning of this section can be provided by several means. One may wonder why this particular design was chosen over others. Below we clarify the reasoning behind the design decisions we took.

a) Replay counter: An attacker may request a certificate from the OEM, and unlock the device. Subsequently, nothing will stop the attacker from running code that issues additional calls to QUERYSTATEANDDEK, even though calls to this method would normally originate only from the bootloader. Providing a previous version of the encrypted $\{\text{DEK}\}$, including one corresponding to a locked state, allows the attacker to learn its plaintext value. Therefore, a primitive is required that allows one to invalidate previous states. Reasoning from the perspective that the storage medium's data integrity cannot be relied upon, some other means of trusted bookkeeping is needed. The simplest primitive that fits the requirement is a counter value. Building it on eFuses, as proposed in Section V-B, prevents even the trusted environment from re-enabling previously invalidated states.

b) Encrypted certificate: Once the OEM has determined that the device is eligible for an unlock, it first signs and encrypts the certificate, then sends it to the user. The outer encryption layer exists in order to force the user into issuing a call to UPDATESTATE before being able to use it, thus ensuring counter value c is incremented. In case this step is not enforced, an opportunity exists for an attacker to learn the DEK of a locked device; by obtaining an unencrypted certificate $\{c, \texttt{UNLOCKED}\}_{\text{skOEM}}$ from the OEM that includes the current counter value c, and storing it in the storage medium, omitting a call to UPDATESTATE. In this situation, the device is unlocked and accepts unsigned code, although the DEK provided to the kernel will be invalid. However, attacker-controlled code can issue an additional call to QUERYSTATEANDDEK and pass the previous certificate $\{c, \texttt{LOCKED}\}_{\text{skOEM}}$. A message sequence diagram of this scenario is given in Fig. 3.

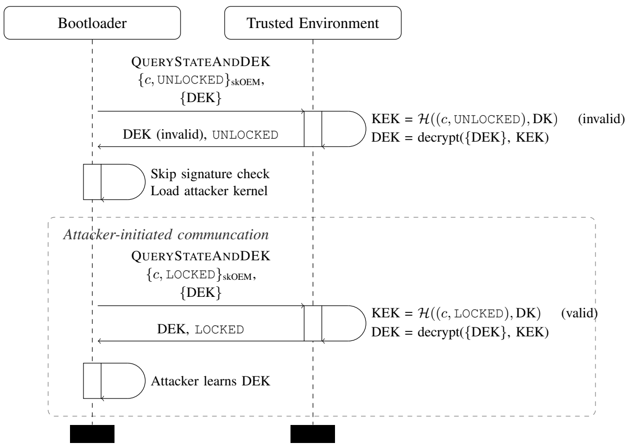

Bootloader
Trusted Environment
QUERYSTATEANDDEK
$\{c, \texttt{UNLOCKED}\}_{\text{skOEM}}$,
$\{\text{DEK}\}$
KEK = $\mathcal{H}((c, \texttt{UNLOCKED}), \text{DK})$ (invalid)
DEK = decrypt($\{\text{DEK}\}$, KEK)
DEK (invalid), UNLOCKED
Skip signature check
Load attacker kernel
Attacker-initiated communcation
QUERYSTATEANDDEK
$\{c, \texttt{LOCKED}\}_{\text{skOEM}}$,
$\{\text{DEK}\}$
KEK = $\mathcal{H}((c, \texttt{LOCKED}), \text{DK})$ (valid)
DEK = decrypt($\{\text{DEK}\}$, KEK)
DEK, LOCKED
Attacker learns DEK

Fig. 3. Possible DEK recovery attack in case unencrypted certificates are used.

c) Multiple calls to QUERYSTATEANDDEK*:* A call to QUERYSTATEANDDEK should only occur once for each time the device is powered on and should originate from the bootloader. Preventing multiple calls from being issued effectively mitigates many potential attacks. However, one should take into account that, when doing so, one becomes dependent on the trusted environment's volatile state. This implies that the trusted environment should never perform a reboot independently of the untrusted environment. This property is difficult to guarantee in practice; e.g., a *Power Management Unit (PMU)* chip may cause it to be violated (see section IV-A0a). Alternatively, an attacker may attempt to crash the trusted operating system, potentially facilitated by physical access. Instead, we accept that multiple calls to QUERYSTATEANDDEK can occur and opt for a design that is secure, regardless of this fact.

E. Formal verification

We used Tamarin, an automatic cryptographic protocol verifier [26], to validate the properties (i), (iv) and (v) listed in section V-A. As is usual for verification of security protocols, the Dolev-Yao attacker model is assumed. Thus, the attacker has complete control over the network but is unable to break cryptography [27]. In the context of this paper, this means that the attacker has complete control over the storage medium, and is able to issue calls to trusted environment procedures at will with arbitrary parameters.

In order to successfully prove the desired properties, the attacker model has to be slightly weakened. Specifically, suppose that the device is re-locked, and thus its DEK is preserved, and that the attacker obtained it while the device was still unlocked. This implies that either:

(i) The attacker himself re-locked the device, thus the attacker already had the opportunity to exfiltrate the user data while the device was unlocked.

(ii) The benign user re-locked the device and consciously passed the DEK to the attacker.

Both scenarios are not relevant for the security of the user data once the device is locked, and can thus be safely ignored.

We modeled the architecture in both Tamarin's native representation, and in *Stateful Applied PI Calculus (SAPIC)* [28], the models are publicly available [29]. For both models, we verified that the device cannot be unlocked without the OEM's approval, that the DEK cannot be obtained while the device is in a locked state, and that a previous variant of the DEK cannot be recovered once the device has been locked. These proofs give confidence that the proposed architecture actually provides the desired properties.

F. Discussion

The proposed architecture provides all the properties listed in Section V-A. However, it does heavily rely on the security of TrustZone and its OS. This has shown to be a less-than-ideal strategy in the past (Section IV). However, we argue that, in order to issue calls into the trusted environment, the attacker needs kernel level privileges, thus the DEK, and hence the user data, is already compromised. Furthermore, even with a full TrustZone compromise, the chain-of-trust still cannot be broken, since it relies on a cryptographic signature that only the OEM holds the private key for. Hence, any security breach is fundamentally temporary until the device is rebooted. As happened with many of the described vulnerabilities, our proposed architecture itself could be subject to implementation flaws. One possible mitigation could be for OEMs to publish their code and ask for security reviews to be performed by independent security experts, together with a reproducible building model to ensure that the audited code matches with the binaries that are shipped to the end users.

Within the assumed attacker model, we have eliminated all angles for obtaining the DEK. An unlock request for the device may be granted by the OEM, but that implies destruction of the DEK. To the best of our knowledge, the only means for obtaining the DEK is either through fault injection (out of scope), or a privilege escalation exploit. The latter is unlikely, since we assume the attacker has no knowledge of the unlock code or pattern, which is required in order to install applications. Hence, in addition to the privilege escalation exploit, the attacker also has to bypass the locking mechanism.

A possible drawback of the proposed architecture is the suggested implementation of the replay counter based on eFuses. This imposes a theoretical limit on the total number of times a device can be unlocked and re-locked. However, low-cost commercial devices nowadays embed a high count of efuses, like 128Kbit in the case of the Kendryte K210 RISC-V SoC [30]. Since every unlock attempt needs an explicit OEM authorization, eFuse exhaustion attacks can be easily hampered by rate-limiting. Finally, a number of lock-unlock cycles in the order of tens of thousands, in practice, hardly represents a limit to the user's freedom.

VI. Concluding Remarks

In this work we analyzed several vulnerabilities related to custom implementation of bootloader unlock procedures and state storage, establishing common pitfalls that cause them. From the highlighted pitfalls we proposed a secure device state storage architecture, which achieves much stronger guarantees. Most notably, it lets OEM have full, cryptographically enforced, control over the unlock authorization. Furthermore, it eliminates the need to trust the storage medium, and finally, a security downgrade enforces a full wipe of the user data by design. We verified the above properties using Tamarin. Finally, the architecture is built on widely available primitives, enabling the deployment on existing devices.

References

[1] G. LLC, "Implementing dm-verity," Mountain View, CA, Tech. Rep., 2019. [Online]. Available: https://source.android.com/security/verifiedboot/dm-verity

[2] S. Tolvanen. (2016) Strictly enforced verified boot with error correction. [Online]. Available: https://android-developers.googleblog.com/2016/07/strictly-enforced-verified-boot-with.html

[3] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983. [Online]. Available: https://doi.org/10.1109/TIT.1983.1056650

[4] J. S. S. T. Association, *Embedded Multi-Media Card (e•MMC) Electrical Standard (5.0)*, June 2012. [Online]. Available: https://github.com/jaehyek/emmc/raw/master/JESD84-B50.pdf

[5] ——, "UNIVERSAL FLASH STORAGE (UFS), Version 3.1," Tech. Rep., January 2020. [Online]. Available: https://www.jedec.org/document_search?search_api_views_fulltext=jesd220e%20ufs

[6] T. L. Project. (2019) Qcom's chain of trust. [Online]. Available: https://lineageos.org/engineering/Qualcomm-Firmware/

[7] G. LLC, "Android verified boot," Mountain View, CA, Tech. Rep., 2019. [Online]. Available: https://source.android.com/security/verifiedboot/avb

[8] T. L. Project. (2019, March) Lineageos android distribution. [Online]. Available: https://lineageos.org/

[9] postmarketOS.org contributors. (2019) A real linux distribution for phones and other mobile devices. [Online]. Available: https://postmarketos.org/

[10] S. Beaupre, "Samdunk: emmc backdoor leading to bootloader unlock on samsung galaxy devices," Tech. Rep., March 2016. [Online]. Available: http://theroot.ninja/disclosures/SAMDUNK_1.0-03262016.pdf

[11] O. Avraham. (2018) Samsung gt-i9300 emmc toolbox. [Online]. Available: https://github.com/oranav/i9300_emmc_toolbox

[12] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Bootstomp: On the security of bootloaders in mobile devices," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 781–798. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/redini

[13] R. Hay, "fastboot oem vuln: Android bootloader vulnerabilities in vendor customizations," in *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017.*, W. Enck and C. Mulliner, Eds. USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/hay

[14] N. Keltner. (2014, August) Here be dragons: Vulnerabilities in trustzone. [Online]. Available: https://atredispartners.blogspot.com/2014/08/here-be-dragons-vulnerabilities-in.html

[15] scotty2, "[S-OFF] revone - DEVELOPER EARLY ACCESS PREVIEW EDITION," https://www.thinkthinkdo.com/trac/project1/wiki/gfree, 2016.

[16] ieftm, "[S-OFF] revone - DEVELOPER EARLY ACCESS PREVIEW EDITION," https://forum.xda-developers.com/t/s-off-revone-developer-early-access-preview-edition.2314582/, Jun. 2013.

[17] S. Beaupre. (2015, February) Firewater is officially discontinued, please read. [Online]. Available: http://firewater-soff.com/

[18] S. M. S. LLC. (2014) Sunshine: bootloader unlock / s-off. [Online]. Available: http://theroot.ninja/

[19] D. Rosenberg, "Unlocking the motorola bootloader," *Azimuth Security Blog*, 2013.

[20] ——, "Qsee trustzone kernel integer over flow vulnerability," in *Black Hat conference*, 2014, p. 26.

[21] laginimaineb, "Full trustzone exploit for msm8974," *Bits, Please!*, 2015.

[22] G. A. Project. (2019) Aosp security features: Device state. [Online]. Available: https://source.android.com/security/verifiedboot/device-state

[23] lowRISC. (2019) Open source silicon root of trust (rot) — opentitan. [Online]. Available: https://opentitan.org/

[24] J. Woodruff, R. N. M. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. M. Norton, and M. Roe, "The CHERI capability model: Revisiting RISC in an age of risk," in *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 457–468. [Online]. Available: https://doi.org/10.1109/ISCA.2014.6853201

[25] D. Barnetson, "Risc-v security: Arm® trustzone® technology vs risc-v multizone™ security," Hex-Five Security, Inc., Redwood Shores, CA, Tech. Rep., March 2019. [Online]. Available: https://content.riscv.org/wp-content/uploads/2019/03/15.05-RISC-V-Security-Multizone-v-TrustZone-3-12-19.pdf

[26] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

[27] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[28] M. Arapinis, J. Liu, E. Ritter, and M. Ryan, "Stateful applied pi calculus," in *International Conference on Principles of Security and Trust*. Springer, 2014, pp. 22–41.

[29] C. Meijer and N. Izzo, "Formal proofs of security of the "Mobile Systems Secure State Management" paper." https://github.com/n1zzo/mobile_systems_secure_state_management, Jul. 2022.

[30] C. Creative, "KENDRYTE DOCUMENT PROJECT," Tech. Rep., October 2019. [Online]. Available: https://github.com/kendryte/kendryte-doc-datasheet

CHAPTER 6

Annexes

Compute Express Link Security in Multitenant Scenarios

Paolo Amato, Alessandro Barenghi, Luca Oddone Breveglieri, Niccolò Izzo, Gerardo Pelosi, Member, IEEE

Virtualization is a key enabler for an efficient and scalable use of datacenter computing resources. Traditionally such flexibility came with a cost in security, as cloud service customers had to blindly trust the service provider with their data. In fact the hypervisor historically retained visibility and control over the data processed by Virtual Machines, furthermore no strong security architectures prevented Virtual Machines to interfere with one another, and security misconfigurations or access to shared peripherals could be exploited by evildoers. Lastly, the Rowhammer vulnerability is a relevant threat in the cloud environment as all the Virtual Machines hosted by an Hypervisor share the same main memory subsystem, thus could leverage the Rowhammer phenomenon to corrupt the data of other Virtual Machines. Even though commercial solutions to exclude the cloud service provider from accessing data exist, they fall short in guaranteeing the extensibility to Compute eXpress Link attached memories, as well as in some cases fail to defend against Rowhammer attacks. In this work we propose and evaluate a security architecture that lets the cloud customer retain exclusive access to its data. Additionally, when the proposed architecture is applied on standard DRAM, the Rowhammer phenomenon is effectively prevented. At the same time, our solution allows the service provider to make use of the extension capabilities of CXL memories, paving the way for future emerging memory technologies, which promise to provide fast, byte-addressable and persistent memory capabilities that can scale to high capacity at reduced cost.

Index Terms—Computer Security, Compute Express Link, Trusted Virtual Machines, Emerging Memories

I. INTRODUCTION

THE advent of emerging memory technology, such as Phase Change Memory (PCM) [1], 3D XPoint (3DXP) [2], Spin-Transfer Torque RAM (STTRAM) [3], and Memristors [4] fills the gap between DRAM-based main memory and NAND-based storage devices. Emerging memories offer a fast, byte-addressable, persistent and scalable memory medium, which can be used as main memory expander, storage accelerator, or by leveraging the persistent memory semantic programming model. To fully exploit the large capacity of emerging memories, and to pave the way for future near-memory processing [5] scenarios, a hot-plug-able cache coherent CPU interconnect is required. The CXL protocol [6], that stems from PCIe 5.0 [7], adding byte-addressable memory operations, fulfills that need by inheriting the flexibility of the PCIe 5.0 PHY and offering a low-latency cache-coherent memory protocol.

Virtualization describes a set of techniques, whose aim is to abstract from the physical hardware, and execute several user Virtual Machines (VM), each running its own operating system on its set of virtualized hardware, on a single physical machine. A Virtual Machine Manager (VMM) or hypervisor is the hardware/software component that is in charge of the hardware virtualization. A VM snapshot can be created to allow easy data back-up; VMs can be migrated between hypervisors, enlarged or shrinked in memory and disk sizes, thus they are a key component to achieve an efficient usage of datacenter resources, and to fit the scalability requirements of the enterprise cloud end-users. However virtualization introduces several security concerns: VMM are designed to minimize the interference between virtual Machines [8], but the VMM itself, who is typically controlled by the cloud service provider, owns and can normally access the main memory of all the virtual machines, and consequently, the data being computed therein.

Traditionally the VMM has been considered a trusted component, however its large codebase and the consequently large attack surface makes it the primary target for privilege escalation in VM to VM and VM to VMM attacks, with more than 10 critical vulnerabilities in the last 3 years [9]. However emerging scenarios such as the hosting of digital voting platforms, where the democratic rights are at stake, or a vaccination certification platform, where massive amounts of sensitive data are hosted, require a higher security grade. Solutions to exclude the VMM from the trusted components are commercially available, the two main ones being AMD SEV-SNP [10] and Intel TDX [11]. VMs hosted in such systems are denominated Trusted Virtual Machines (TVM). They work by encrypting and decrypting data at the memory controller level with a key that is bound to the software entity in execution, e.g., a VM or the VMM; such keys are not accessible to the VMM and prevent the readout, and tampering of data contained in the main memory of a Virtual Machine.

Nonetheless, address space which is mapped to peripherals, including CXL devices, cannot be encrypted under AMD-SEV and Intel TDX, which recommend as an alternative to employ E2E encryption strategies [10]. However, performing E2E encryption in software leads to

This paragraph of the first footnote will contain the date on which you submitted your paper for review. It will also contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant BS123456".

P. Amato is with Micron Technology, Inc. (e-mail: pamato@micron.com).

A. Barenghi, L. Breveglieri, N. Izzo and G. Pelosi are with Politecnico di Milano, Milan, Italy. (e-mail: name.surname@polimi.it).

performance degradation, due to the necessity of using bounce buffers, furthermore, the peripheral might need to access data, e.g. to perform near-memory computation.

Finally emerging memory, with its data persisency features requires encryption and integrity capabilities as data is retained even after a power-loss event, facilitating all the cold-boot-like attack scenarios.

A. Contribution

We propose a security architecture, in which the hypervisor retains resource management but is no longer trusted for securing traffic towards peripherals, including PCIe devices, CXL devices such as emerging memory-based accelerators. The proposed architecture is able to reduce the trust boundary, allowing secure and fast data transfers between VMs and peripherals, encrypting data on the host side, to guarantee data confidentiality and integrity even in the case of persistent emerging memory applications. Finally, we present an evaluation of the performance cost of the proposed solution.

II. Background

CXL [6] is an open industry standard for interconnecting CPUs, accelerators, and memory devices, designed to achieve high-bandwidth and low latency. CXL is based on the PCIe 5.0 [7] PHY, however, while PCIe is oriented towards non-coherent DMA transfers; CXL is designed for cache coherent 64B granularity byte-addressable transfers. CXL can reach 128GB/s of bandwidth when configured in x16 lanes mode, and has an expected latency in the order of the hundreds of nanoseconds [12]. The CXL protocol is currently being developed by the CXL Consortium, in collaboration with several companies in the datacenter ecosystem. CXL is composed of three different sub-protocols: CXL.io for discovery, configuration and non-coherent memory transfers, CXL.mem for cache-coherent memory transfers and CXL.cache for managing cache semantics. The three protocols are multiplexed on a single CXL link, that is established on the PCIe 5.0 PHY. Presently, the CXL protocol describes three types of devices: CXL Type 1 devices are computation accelerators that implement the cache-coherent memory interface without having any on-board main memory; CXL Type 2 devices are accelerators that are equipped with on-board cache-coherent main memory; and CXL Type 3 devices are main memory expanders, as they feature only an external cache-coherent main memory.

A VM whose main memory is not accessible by the VMM hosting it, is commonly denominated as Trusted Virtual Machine (TVM). The main commercial solutions to obtain TVMs are Intel Trust Domain Extensions (TDX) [11] or AMD Secure Encrypted Virtualization (SEV) [10]. These solutions work by encrypting data before it is stored in the main memory, and decrypting data when it is read from the main memory. The encryption is performed by a hardware engine embedded in the main memory controller, which selects a different encryption key for each software entity in execution in the system, i.e. different VMs, TVMs, or the VMM itself. The management of the encryption keys is delegated to a separate CPU which in the case of AMD SEV-SNP is called AMD Secure Processor (SP), in the case of Intel TDX is called Intel Management Engine (ME). In all the cases, the entity managing the encryption keys is not under the control of the VMM, and is considered trusted.

In a virtualized setup, a cloud provider might need to share a single CXL Memory expander between multiple VMs. The CXL 2.0 standard defines a Multi Logical Device (MLD) functionality, which allows to partition a physical device into up to 16 Logical Devices (LD). LDs are visible to the Virtual Hierarchy (VH) and share transaction and link layers.

In the past several memory encryption techniques have been proposed, they can be partitioned according to whether the encryption process is performed on the host or on the device, Host-Side Encryption (HSE) include AMD SEV-SNP [10] and Intel TDX [11], while Device-Side Encryption was generally adopted for Self-Encrypting Drives, and TCG Opal [13]. The advantages of Device-Side Encryption include the delegation of the encryption cost, the enablement of encrypted Peer-to-peer communication scenarios, the ease of upgradeability to new ciphers by replacing just the peripheral. But DSE requires the peripheral or part of it to be fully trusted by the Trusted Virtual Machine. Several solutions try to increase the trustworthiness of peripheral components such as Microsoft Cerberus [14] and TCG DICE [15], but trusting peripherals manufactured by several third-party companies is a critical step in the enlargement of the Trust Boundary. On the other side Host-Side Encryption greatly simplifies the key management process and allows to consider the peripheral as semi-trusted, that is trusted only for the availability of resources, keeping the Trust Boundary small.

III. Threat Model

The primary application for the CXL protocol will be in the hyperscaler datacenter environment. A datacenter might be thought as a tightly physically protected environment, nonetheless the possibility of insider threats, the presence of third party service contractors, and the need for independence from the cloud-provider itself, pose the need for the formalization of an ad-hoc threat model.

The goal of the proposed architecture is to guarantee on the data stored in a CXL type 3 memory device, assigned to a Trusted Virtual Machine, the following properties:

- Data Confidentiality: data should not be accessible outside the TVM trust boundary.
- Data Integrity: data should not be modifiable outside the TVM trust boundary, not even indirectly, e.g. through a Rowhammer attack.
- Data Replay Protection: data should not be replaceable with an older copy.

The architecture relies on existing solutions for the encryption and integrity protection of the DDR-attached main memory, therefore protection of such memory is considered out-of-scope in this work.

The direct access virtualization model must not require intervention of software outside the TVM Trusted Computing Base (TCB) to perform operations that affect the confidentiality and/or integrity of the TVM data that is in-flight or at-rest in the device.

Protection against denial of service is left out-of-scope as hypervisor is considered semi-trusted, as in trusted only regarding resource management

The security architecture must guarantee the following security properties:

We provide a ranking of several threat scenarios, from now on termed Threat Levels (TL), from the most contrained one, to the one with most capabilities. The set of capabilities of every Threat Level is a superset of the capabilities of the Threat Levels with lower index.

TL1 In this Threat Level, the attacker can only execute software on the same host, either in a separate VM or in the VMM. We assume a completely white-box scenario with respect to virtual-to-physical memory mapping, cache and memory geometry. In fact these details could be either publicly available for some platforms, already have been reverse-engineered [16], or could be reverse-engineered with documented techniques [17], [18]. Given the availability of cache geometry and memory mapping information, this Threat Level allows an attacker to mount Rowhammer attacks.

TL2 In this Threat Level the attacker has physical access to the server on which the TVMs are in execution, and is able to rework Surface Mount Tecnhology (SMT) components, and insert interposers on all the system buses. However the attacker limits itself to passive attacks which involve recording and analyzing logic signals. Examples of such attacks are Side Channel Attacks based on the accessed memory addresses.

TL3 Attackers in this Threat Level have the same capabilities of TL2, but they can in addition perform active attacks that entail the manipulation of electric signals, including the injection of rogue bus transactions and the replay of old bus transactions.

TL4 This Threat Level embeds all of the above capabilities, with also the capability of performing sophisticated Nation-State level attacks that go beyond the chip package level, including microprobing, 2D/3D X-Ray scans, Near Infra-Red (NIR) imaging, inverse microscopy, Focused Ion Beam (FIB) editing.

Achieving protection against the described Threat Levels will have a cost that is proportional to the capabilities of the highest mitigated Threat Level.

IV. Architecture

The proposed architecture implements a Host-Side Encryption process, that enables Trusted Virtual Machines to benefit from a CXL type 3 memory expander without enlarging their Trust Boundary.

A. System Integration

We leverage a component called Memory Encryption Engine (MEE) to align with the nomenclature of other solutions from the state of the art [19]. The MEE sits in-line on the main memory datapath, and for each memory transaction examines the identity of the originating process that can be a non-trusted entity, e.g. the UEFI, the VMM, a traditional VM, or a trusted entity, that is a TVM. Information about the identity of the process is already available at the memory controller level as it is already adopted to build architectures such as AMD SEV or Intel TDX. The identity of the originating software can be encoded as an integer that is biunivocally associated with each software entity, that we will define Entity IDentifier (EID). The EID is assigned by the Memory and Coherence Logic (MCL) handling the memory transactions and cannot be read or written by the software in execution on the main CPU.

For each memory transaction, the MEE uses the EID to index a Ternary Content Addressable Memory (TCAM) that holds the protection settings for each entity, as well as the encryption and integrity keys. According to the protection settings specified in the TCAM, the MEE either passes through the memory transaction without encryption and integrity protection, or it uses the key loaded from the TCAM to perform and encryption (for memory writes) or decryption (for memory reads) operation.

We assume the lifetime of the memory encryption and integrity keys to be shorted than the time of operation of the device, a cold system boot requires the re-generation of fresh keys. This principle is naturally enforced by the TCAM being a volatile data structure.

B. Memory Protection Levels

The algorithmic choice to be implemented depends on the Threat Levels which needs to be successfully mitigated. Performing encryption of the TVM memory is sufficient to thwart TL1 and TL2, as any other entity, even if capable of reading inside the address space of the target TVM, would not be able to read the content stored inside that memory region. Rowhammer attacks, which are entailed by TL1 can enable the undetected corruption of the plaintext, in a way that depends on the chosen cipher and mode of operation. A non-malleable cipher is recommended as it prevents precise bit-level corruption of the plaintext, lowering the chances of a successful attack targeting memory corruption. If the cipher is non-malleable, the attacker can only corrupt one or more blocks of the cipher, as a consequence, if the data structure to be protected is equipped even with a simple Cyclic Redundancy Check (CRC), the attacker needs to generate a corrupted block that features a valid CRC, thus needs to rely on CRC collision, which can be made arbitrarily unlikely to happen by controlling the bit length of the CRC.

An integrity protection mechanism can be employed to avoid plaintext corruption, even at block-size level. The

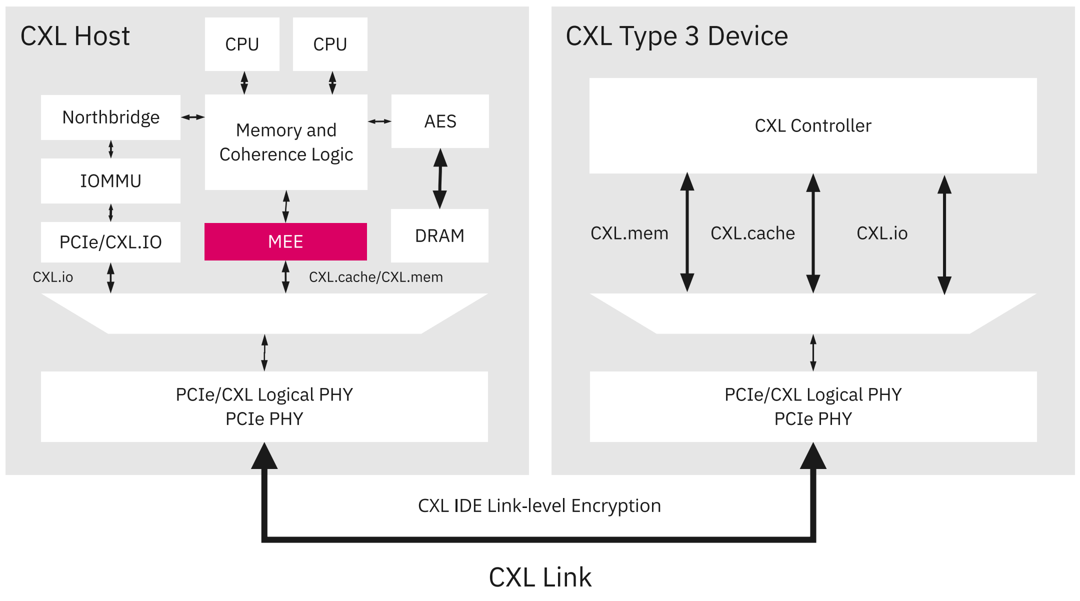


Fig. 1. Block diagram representation of the proposed security architecture. The block diagram includes the memory subsystem of a CXL-enabled CPU, attached via CXL bus to a CXL memory expander device. In magenta is highlighted the Memory Encryption Engine (MEE), which is used to encrypt the data to be stored in the CXL peripheral. The CXL link itself is secured by the CXL IDE Link-level encryption.

CXL protocol, among its metadata, includes a poison bit [12], which is set when an error occurs during data read. A data integrity system needs to set the poison bit whenever during a read operation the integrity check fails, thus the system will be able to distinguish data tampering or corruption attempts and will not consume the corrupted data. Independent encryption and integrity algorithms could be adopted, or an Authenticated Encryption (AE) algorithm, able to achieve both confidentiality and integrity on the TVM data could be used.

Attackers in both TL1 and TL2 can obtain access to the sequence of addresses read or written by a victim TVM, either through the exploitation of cache timing side channel [20] (TL1) or directly by monitoring the address data lines of the memory bus (TL2). The sequence of accessed addresses is considered sensitive in some cases, including the execution of software ciphers where the address sequence might enable a cryptographic side channel attacks by being correlated with the input or output of an S-Box (in lookup table implementations) [21], or the execution of search algorithms over confidential data under complete white-box assumptions, that is where the attacker knows the search algorithm [22]. We consider the protection of the confidentiality of the addresses to be out-of-scope in this work. Solution for this problem have already been proposed in the form of variations of the Oblivious Random Access Memory construction [23].

To achieve resistance under the Threat Level 3, the cryptosystem needs to be able to distinguish subsequent versions of a ciphertext from the previous versions. This can be achieved by adopting a mode of operation that accepts an Initialization Vector (IV) or nonce, that is a bit array that influences the ciphertext and need to be provided during the decryption as well to preserve the data. The IV needs to be changed every time a memory encryption block is updated, providing what is identified in the state of the art as freshness [19]. Many cryptosystems such as AES-GCM [24] as not nonce-misuse resistant, in fact if two different ciphertext are encrypted using AES-GCM with the same key and the same IV, the attacker can recover the plaintext and forge new ciphertexts [25], [26]. Randomizing the IV at each re-encryption requires a source of entropy and would expose the system to the possibility of IV collisions. Instead a counter is typically used, initialized to 0 when a new key is employed, and incremented at each memory write. Counter overflows must be prevented and should trigger a key rotation of the whole encryption block. The counter itself, as it is needed for the decryption, must be stored, e.g. alongside the encrypted data, and even though is not subject to confidentiality requirements, must be secured against modifications by evildoers.

To ensure that the counters are not rolled back by an attacker with TL3 capabilities, a hash of all the memory counters need to be performed at each read operation. To minimize the computation effort for hashing a large amount of data, a Merkle Tree [27] can be employed. A Merkle Tree, as depicted in Fig. 2, performs the hashing operations in a hierarchical way, all the counters of the integrity protected memory blocks are considered as the

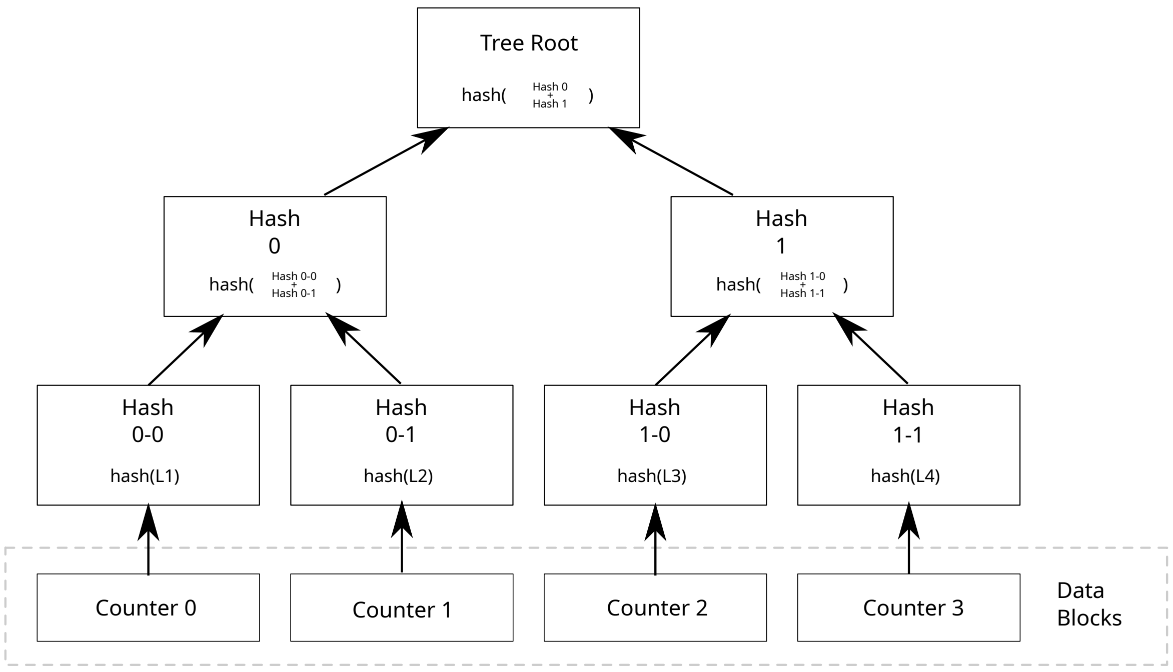


Fig. 2. Block diagram representation of a Merkle Tree. The leafs of the tree are the counters serving as IVs for the selected cryptosystem, counters are hashed together in groups of n to form the upper layer of the Hash Tree. The tree has a single root, which needs to be stored inside the Trust Boundary as it needs to be integrity protected.

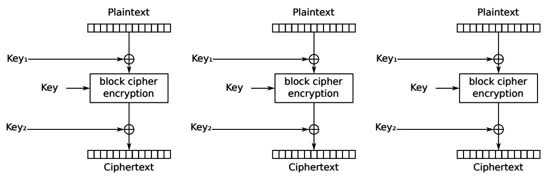


Fig. 3. XOR-Encrypt-XOR (XEX) mode encryption

leaf of a n-ary tree. Every time a leaf is updated, the branch that leads that leaf to the root needs to be re-computed as well, including the root node. The root node should be stored in a secure register inside the Trust Boundary. The arity of the tree can act as a trade-off between the height of the tree and the computation time of each hash, and needs to be tuned to increase the efficiency of the security architecture.

Attackers with capabilities in TL4 require the introduction of anti-tampering on-chip countermeasures and are left out-of-scope in this work.

C. Algorithmic Choice

Among the many possible algorithmic choices to perform confidentiality and integrity protection, we offer two choices based on the AES block cipher, and three choices based on emerging lightweight ciphers.

AES is a 128-bit block cipher, as such it needs to be embedded in a suitable mode of operation to be applied on data which is larger than 128-bit, such as the 64B CXL cache-line. We selected the AES XOR-Encrypt-XOR mode of operation [28] (shown in Fig. 4) due to its non-malleability property and it's high parallelization features. The AES-XTX cipher alone only guarantees confidentiality, therefore we need to apply a keyed hash function on the data as well to protect the data integrity. We select one hash function among the family of the Universal Hash Functions (UHF) [29] that allow a better parallelizability. The keyed hash is computed on the ciphertext to allow the parallel execution of the decryption and hashing during memory reads, that could help improve the read latency.

$$\text{AES-XEX}(IV, K, ptx) = ctx; \text{UHF}(K, ctx) = tag$$

One could also choose an AES mode of operation that performs Authenticated Encryption, such as Galois Counter Mode (GCM) [25], such mode of operation allows to encrypt the data and compute an integrity tag in one pass. AES-GCM is not nonce-misuse resistant, therefore this mode of operation required freshness of the IV, that can be accomplished by using a counter as IV, incrementing it at each memory write, and rotating the encryption key, performing a full re-encryption of data when a counter reaches its maximum value.

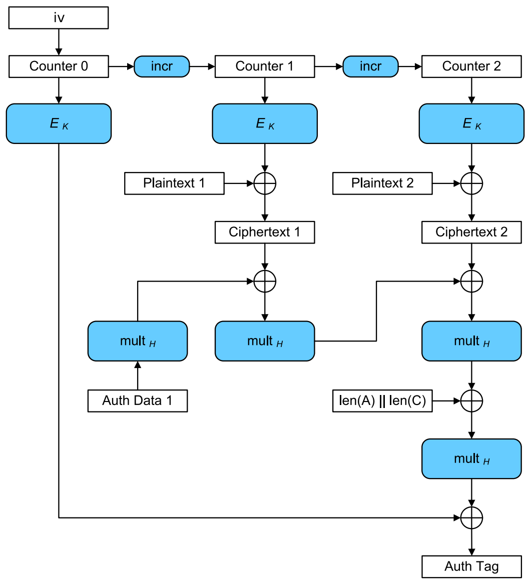


Fig. 4. Galois Counter Mode (GCM) mode encryption

$$\text{AES-GCM}(IV, K, ptx) = ctx, tag$$

Moving away from the AES block cipher, in the latest years, several Authenticated Encryption with Associated Data (AEAD) ciphers where proposed. One of these is QARMA [30] by Avanzi et. al., which has been employed in several main memory encryption strategies.

$$\text{QARMA}(IV, K, ptx) = ctx, tag$$

From 2012 to 2019, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) evaluated AEAD proposals for constrained resource, high-performance, and defense in depth applications. The selected AEAD cipher for high-performance applications is AEGIS-128.

$$\text{AEGIS-128}(IV, K, ptx) = ctx, tag$$

Another AEAD cipher that is worth mentioning is called KETJE-jr, and is based on the cryptographic sponge structure adopted in the recently standardized SHA-3 hashing algorithm, albeit with reduced numer of round and reduced state size.

$$\text{KETJE-jr}(IV, K, ptx) = ctx, tag$$

We will evaluate the overhead and performance of the proposed cryptographic solutions, however the ultimate choice depends on the availability of silicon area in the target application.

V. Methods and Procedures

We propose an analytic evaluation of the overhead of the proposed protection schemes, evaluated on the available bandwidth of a CXL 2.0 x16 link.

VI. Results

VII. Conclusion

References

[1] A. Redaelli. Phase Change Memory: Device Physics, Reliability and Applications. Springer International Publishing, 2017.
[2] Micron. Breakthrough nonvolatile memory technology, 2018.
[3] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. In IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest., pages 459–462, 2005.
[4] Dmitri B. Strukov, Snider S. Gregory, Stewart R. Duncan, and Stanley R. Williams. The missing memristor found. In Nature. Nature, 2008.
[5] Adrián Barredo, Adrià Armejach, Jonathan C. Beard, and Miquel Moretó. PLANAR: a programmable accelerator for near-memory data rearrangement. In Huiyang Zhou, Jose Moreira, Frank Mueller, and Yoav Etsion, editors, ICS '21: 2021 International Conference on Supercomputing, Virtual Event, USA, June 14-17, 2021, pages 164–176. ACM, 2021.
[6] Dr. Debendra Das Sharma and Siamak Tavallaei. Compute Express Link™ 2.0 White Paper. Technical report, CXL Consortium, November 2021.
[7] PCI-SIG. PCI Express Base Specification Revision 5.0, Version 1.0. Technical report, PCI-SIG, May 2019.
[8] The QEMU Project Developers. Security, 2021.
[9] MITRE Corporation. Qemu : Security vulnerabilities (cvss score >= 7), 2021. Last accessed on 09.12.2021.
[10] Advanced Micro Devices, Inc. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. Technical report, January 2020.
[11] Intel Corporation. Intel® Trust Domain Extensions. Technical report, Intel Corporation, 2021.
[12] Compute Express Link Specification. Compute Express Link Specification, 2021. Last accessed on 09.12.2022.
[13] Trusted Computing Group. TCG Storage Security Subsystem Class: Opal Specification. Technical report, Trusted Computing Group, 2022.
[14] Bryan Kelly. Project Cerberus Security Architecture Overview Specification. Technical report, Open Compute Project, 2019.
[15] Trusted Computing Group. DICE Attestation Architecture. Technical report, Trusted Computing Group, 2021.
[16] Mark Seaborn. L3 Cache Mapping on Sandy Bridge CPUs, 2017. Last accessed on 11.22.2022.
[17] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. CoRR, abs/1507.06955, 2015.
[18] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-only reverse engineering of physical DRAM mappings for rowhammer attacks. In 3rd IEEE International Verification and Security Workshop, IVSW 2018, Costa Brava, Spain, July 2-4, 2018, pages 19–24. IEEE, 2018.
[19] Roberto Avanzi. Cryptographic protection of random access memory: How inconspicuous can hardening against the most powerful adversaries be? In Francesco Regazzoni and Marten van Dijk, editors, Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW 2022, Los Angeles, CA, USA, 7 November 2022, page 1. ACM, 2022.
[20] Younis A. Younis, Kashif Kifayat, Qi Shi, and Bob Askwith. A new prime and probe cache side-channel attack for cloud computing. In Yulei Wu, Geyong Min, Nektarios Georgalas, Jia Hu, Luigi Atzori, Xiaolong Jin, Stephen A. Jarvis, Lei (Chris) Liu, and Ramón Agüero Calvo, editors, 15th IEEE International Conference on Computer and Information Technology, CIT 2015; 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015; 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015, Liverpool, United Kingdom, October 26-28, 2015, pages 1718–1724. IEEE, 2015.
[21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. IACR Cryptol. ePrint Arch., page 271, 2005.

[22] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: abusing intel SGX to conceal cache attacks. Cybersecur., 3(1):2, 2020.

[23] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In Alfred V. Aho, editor, Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA, pages 182–194. ACM, 1987.

[24] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). IACR Cryptol. ePrint Arch., page 193, 2004.

[25] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, Information Technology Laboratory, National Institute of Standards and Technology, 2007.

[26] birb007. AES-GCM Misuse, 2022. Last accessed on 11.22.2022.

[27] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings, volume 293 of Lecture Notes in Computer Science, pages 369–378. Springer, 1987.

[28] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings, volume 3329 of Lecture Notes in Computer Science, pages 16–31. Springer, 2004.

[29] Larry Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA, pages 106–112. ACM, 1977.

[30] Roberto Avanzi. The QARMA block cipher family - almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. IACR Cryptol. ePrint Arch., page 444, 2016.

Paolo Amato received laurea degree (cum laude) in computer science from the University of Milano in 1997 and the Ph .D. in computer science from the University of Milano-Bicocca in 2013. He joined Micron in 2010, where he is Distinguished Member of the Technical Staff and the System Architect for Automotive storage solutions. His current focus is on UFS solutions. He is an expert on statistical methods, error correcting codes, and security. From 1998 to 2008, he was with STMicroelectronics (Agrate, Italy). From 2000 to 2005, he led the "Methodology for Complexity" team (a global R&D team of about 30 people located in Milano, Napoli, Catania, Moscow , and Singapore) aimed at developing methods, algorithms, and software tools for complex system management. From 2005 to 2010, he was the Manager of Statistical Methods for the Non-Volatile Memory Technology Development group in ST and Numonyx, and he started the development of ECC solutions for PCM devices. From 2010 to 2018 he has been a System Architect for Mobile storage solutions. From 2018 to 2022 he has been the director of the Memory System Pathfinding team in Vimercate (Italy) focusing on CXL solutions for data center application. He has authored about 50 papers published in peer-reviewed international journals and international conferences. Dr. Amato has filed around 100 patents.

Alessandro Barenghi is an associate professor at Politecnico di Milano, Italy. His interests include computer and network security, formal languages and compilers and he has published more than 80 papers in international peer reviewed venues.

Luca Breveglieri was born in 1962, Italy. He received the M.Sc. degree in electronic engineering and the Ph.D. degree in electronic engineering of information and systems from Politecnico di Milano, Milan, Italy, in 1986 and 1991, respectively. From 1991 to 1998, he worked as an Information and Communications Technology (ICT) manager and network administrator. In 1998, he was appointed an Associate Professor in Politecnico di Milano, in the School of ICT. He researched the design of dedicated architectures for computer arithmetic and signal and image processing. Since he was appointed an Associate Professor, he has been researching mainly the fields of security of digital devices, both dedicated and programmable, of efficient applied cryptographic algorithms, and of the protection against attacks (mathematical and side channel); and secondarily the field of theoretical computer science (formal languages and automata). He has participated in several European Union (EU) and national research projects (MEDEA+, ENIAC and PRIN), mostly on the security of digital devices. He collaborated with CERN, Geneva, on the design of a Digital to Analog converter (DAC) for the particle detectors for Large Hadron Collider (LHC), and with STMicroelectronics and Micron on applied cryptography topics. He is coauthor of over 100 international peer-reviewed journal and conference papers on his research topics. He supervises the supercomputing activities by his university.

Niccolò Izzo is pursuing a Ph.D. degree in information technology, at Politecnico di Milano, Italy. Among his research interests are security in traditional and emerging memory subsystems, cryptographic and microarchitectural side channel attacks. He collaborated with Micron on the application of cryptography on emerging memory devices, co-authoring more than 10 patents.

Gerardo Pelosi is an associate professor at Politecnico di Milano, Italy. His main research interests are in computer security, cryptography, security in hardware and in the area of data security and privacy. He has published more than 90 papers in international peer reviewed journals and conference proceedings and is co-inventor of 10 patents concerning the design of cryptographic systems.

An expanding universe poses an inevitable faith on its inhabitants.
All the bonds will break and everything will fade out into noise.
It’s up to us to locally push back entropy, taking care of our bonds,
Is it our purpose, to retrieve a signal through the noise?