On Trustworthy Scalable Hardware/Software Platform Design

Friedrich Pauls, Sebastian Haas, Stefan Köpsell,
Michael Roitzsch, Nils Asmussen, Gerhard Fettweis
Barkhausen Institut, Dresden, Germany
forename.surname@barkhauseninstitut.org

***Abstract*—The continuously growing importance of today's technology paradigms such as the Internet of Things (IoT) and the new 5G/6G standard open up unique features and opportunities for smart systems and communication devices. Famous examples are edge computing and network slicing. Generational technology upgrades provide unprecedented data rates and processing power. At the same time, these new platforms must address the growing security and privacy requirements of future smart systems. This poses two main challenges concerning the digital processing hardware. First, we need to provide integrated trustworthiness covering hardware, runtime, and the operating system. Whereas integrated means that the hardware must be the basis to support secure runtime and operating system needs under very strict latency constraints. Second, applications of smart systems cover a wide range of requirements where "one-chip-fits-all" cannot be the cost and energy effective way forward. Therefore, we need to be able to provide a scalable hardware solution to cover differing needs in terms of processing resource requirements.**

In this paper, we discuss our research on an integrated design of a secure and scalable hardware platform including a runtime and an operating system. The architecture is built out of composable and preferably simple components that are isolated by default. This allows for the integration of third-party hardware/software without compromising the trusted computing base. The platform approach improves system security and provides a viable basis for trustworthy communication devices.

***Keywords*—Composable systems, Hardware/software co-design, Isolation, Microkernel, Operating system, Privacy, Security, Tiled architecture**

I. Introduction

Smart systems and devices have become an integral part of our lives that we rely on daily. Logistics, healthcare, and the automotive industry are only some examples where smart digital functionalities are key enablers to deploy intelligent systems. Mostly, devices communicate via wireless connections. Especially, 5G/6G mobile networks may provide the required data rate and latency at the same time. Furthermore, these technologies bring many new features such as network slicing and edge computing to the design and implementation of the networks. However, these enhancements also introduce new security and privacy risks.

For example, *network slicing* is a new 5G feature which provides different logical mobile networks on top of the same physical hardware. The concept allows to provision different network slices with different quality of service guarantees in terms of latency, throughput, or reliability without the need to provide dedicated hardware for every network. But it also introduces the risk, that one slice interferes with another slice in an unwanted way if there is no secure and reliable separation of these slices.

Another new feature is *edge computing*. Here the main idea is that the mobile network will not only provide communication capabilities but also computational resources. Moreover, these computational resources will be located close to the user equipment. A dependable separation between user applications is important here as well to prevent data privacy issues.

In order to realize such features, new architectural and implementation paradigms and concepts need to be applied. One of the related 5G pillars is the so-called *softwarization* [1]. The basic idea is to realize functionality in software, which traditionally was implemented in hardware. Softwarization is accompanied by concepts like disaggregation of hardware and software and decomposition of functions into small components with well-defined interfaces [2]. In fact, the idea of softwarization is not specific to mobile networks. Non-mobile networks have already adopted concepts like software-defined networks (SDN) or network function virtualization (NFV). SDN refers to the idea of splitting traditional, monolithic network elements like routers or bridges into a rather dumb data plane which only handles the packet forwarding itself and a more "intelligent" control plane which executes all routing decisions. Thereby, the latter is often implemented in software only. NFV is a concept which uses virtualization to execute the whole (dedicated) functionality of a network node in software.

The 5G mobile network extends these concepts by also softwarizing the radio access network (RAN) as much as possible. This is done with the help of so-called software-defined radios (SDRs), where, in the extreme case, most of the signal processing happens in software on general-purpose processors. Specialized hardware is only required for analog-to-digital converters, digital-to-analog converters, power amplifiers, and antennas.

Although softwarization promises great benefits in terms of flexibility and cost reduction, it comes at the price of a larger attack surface. Therefore, IT security related risks increase tremendously [3]. The attack surface includes all interaction points an unauthorized entity ("attacker") can reach. It covers interfaces, protocols, and services as well as software and hardware.

In order to mitigate these security risks, we propose an

integrated hardware/software co-design for building custom compute hardware from pluggable components and modular system software that is specifically tailored to the aforementioned smart systems and applications. The core of our concept is the decomposition of the overall system into *trusted* components, which we must rely on to enforce a specific security policy, as well as *untrusted* components which are not part of the attack surface anymore because they are isolated. While this concept is already well understood in the context of software, we extend it to cover also the hardware. We believe that this is important because recently discovered vulnerabilities in conventional processors have shown that hardware insecurities are not only theoretic [4]. Furthermore, the modular hardware design allows to provide a scalable solution to cover differing needs in terms of processing resource requirements.

In the following sections, we present our approach on a composable architecture at both the software (Section II) and hardware level (Section III). Based on that, in Section IV we describe how pluggable components at both levels can be combined to build customized and secure platforms for smart systems. In Section V, we present related work, and finally in Section VI, we discuss open research topics where our approach may already provide the basis for practical solutions.

II. Software Components

Attacks on software are commonplace today. Data breaches, machine takeovers for cryptocurrency mining, and ransomware attacks are constantly in the news. Whenever software handles input that an attacker can control, the software processing this data constitutes an attack surface. This situation naturally occurs, when devices are connected and handle input from the Internet. An attacker can maliciously craft an input and thereby exploit a programming error in the software to gain control over it. However, the severity of such a programming error and the resulting leverage of the attacker depends on the software architecture in place. Many of the security problems we see in practice are caused by a lack of isolation between individual components of a software system.

For systems based on a *microcontroller*, all code on the platform runs in the same isolation domain. Microcontrollers lack the architectural features to run an operating system (OS) kernel, because they do not support the division into kernel mode and user mode in which applications are executed. With one globally available memory space, any code in the system can modify any piece of data. Therefore, separation of the system into multiple components is purely a software engineering tool, but not a security barrier. An attacker is just one vulnerability away from full control over the platform.

Monolithic systems improve this situation because they run on hardware featuring an isolated kernel environment. Using memory separation by the memory management unit (MMU), multiple address spaces are implemented, which can only access the memory assigned to them by the kernel. This construction yields much stronger isolation because different concerns of the software system can run in separate processes which are unable to manipulate each other's data. However, the kernel itself still constitutes a single point of failure. Any piece of code in the kernel has full access to the entire system. The huge number of lines of kernel code unintentionally introduce complex bugs into components like file systems, device drivers, and network stacks. One exploit in a network protocol implementation still gains the attacker full control and these exploits are real in monolithic systems.

Microkernel-based systems (e.g. L4 [5]) try to remedy this unfortunate situation by decomposing the kernel itself into multiple, isolated pieces. Drivers, file systems, and protocols no longer run as part of the kernel, but as unprivileged service processes within their own address space. Exploiting a vulnerability in a single component is not sufficient to compromise the whole system. An attacker has to find a chain of exploits spread over multiple components. This inherently increases the security of the system.

Microkernel-based systems are not a panacea. Feature-rich applications and technology stacks still introduce a significant portion of complexity. However, the microkernel-based approach enables to compartmentalize the complexity, such that only a subset of components needs to be relied upon to maintain a specific security objective like confidentiality or integrity. This reliance set of components is called the *trusted computing base* (TCB).

We take the example of the gateway in a virtual private network (VPN) which is implemented with three components on top of a microkernel [6]: one component includes the TCP/IP network stack for the Internet-facing network adapter, a second component provides the same stack for the intranet-facing network adapter, and a third component implements the VPN cryptography layer between the intranet and the Internet. Regarding confidentiality and integrity of the VPN traffic, only the latter two components have access to plaintext and are therefore part of the TCB. Attacks from the Internet on the VPN device would be absorbed by the Internet-facing component, which never has access to plaintext within the confines of its address space. Thus, microkernel-based systems help to reduce the size of the TCB and put up additional isolation barriers to contain attacks. In contrast, in a monolithic system both network interfaces would be driven by the kernel, leaving all components in the TCB without isolation barriers between them.

III. Hardware Components

The software is only as secure as the hardware it runs on. Thus, hardware is always part of the TCB. Our approach tries to minimize the extent to which we need to trust a hardware platform. We see two opposing challenges: From a security perspective, different systems and applications shall be isolated as much as possible, for example, by physically separating them on different servers, so that no interaction is possible in the first place. On the other hand, from a performance, energy, and latency perspective, it is beneficial to integrate systems very closely together. This is especially important in future 5G/6G networks which promise to provide network slicing or mobile edge computing capabilities. These applications

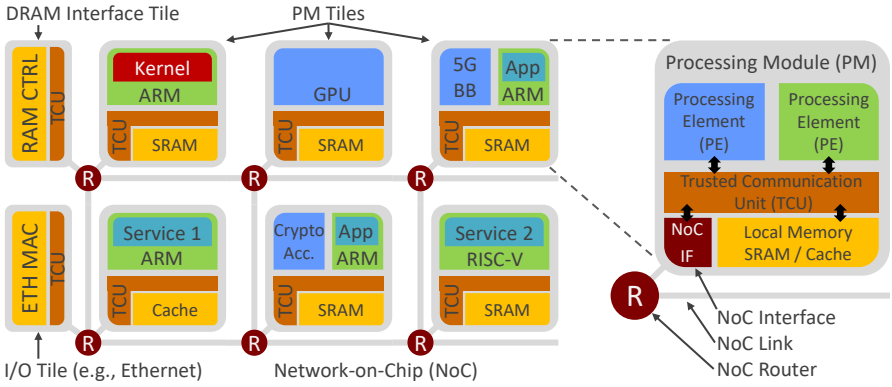

Fig. 1. Our system architecture with multiple tiles isolated by trusted communication units (TCUs)

combine requirements like high-throughput, low-latency, low-power, resiliency, and ultimately the ability to flexibly choose between them depending on the set of applications currently running.

One possible solution which can cope with these requirements are tightly integrated heterogeneous processing platforms, for example, as proposed in Arnold et al. [7]. These architectures offer high-performance and low-power general-purpose and digital signal processors together with specialized hardware accelerators which provide the necessary performance and power efficiency. The hardware components can be flexibly composed into larger applications while the software configures and controls the system.

The envisioned application scenarios do not only require high-performance processing platforms, but also the possibility to run various applications of multiple users in parallel. Multiple network operators might share the same base-station hardware to provide low-latency network services, network-slicing operation, or edge computing. Such an integrated system with multiple users creates safety, security, and privacy problems; two important ones are explained in the following.

One security problem arises from the fact that high-performance general-purpose cores require out-of-order execution and speculative execution to deliver the expected performance. With the discovery of vulnerabilities like Spectre [8] or Meltdown [9], it has been shown that data leaks are possible even across the boundaries of traditional isolation mechanisms. The complexity of these features and their complex interplay are likely the reasons that those vulnerabilities exist in the first place, and that they could be lurking unnoticed in CPUs for many years. While mitigations have been proposed [10, 11], they match the complexity of the features they are trying to secure, and thus make it likely that new security gaps are found in the future. Later, more recent attacks have emerged which are termed *microarchitectural data-sampling* (MDS) [12]. In conventional monolithic operating systems and microkernel-based OSes, the kernel utilizes different privilege levels of the processor hardware to isolate itself from applications and the applications from each other. MDS attacks use these shared processor-intern resources to break the necessary isolation.

Another security issue is a consequence of the complexity of hardware design. System designers often rely on *third-party* hardware components which are designed by others, but still integrated into their systems. These so-called *IP blocks* enable the integration of a variety of processors, accelerators, and interfaces, which are needed to meet the functionality, cost, and time-to-market requirements. However, IP blocks may be part of the TCB although their real content is not known. A single malicious or faulty component may compromise the whole hardware platform, and as a result, all the software running on it.

Both issues mentioned above justify the fact that the software is only as trustworthy as the hardware it runs on. Thus, we must be able to trust the hardware as it is always part of the TCB and plays a key role in the security of the system. A complex hardware design leads to a large TCB which must include different processors, interfaces, and IP blocks from a variety of suppliers. In contrast, our goal is to minimize the TCB. In the next section we describe our proposed hardware architecture and the mechanisms we developed to ensure isolation between different system components. Before, we describe state-of-the-art hardware techniques we build upon to address efficiency requirements.

Modern *multiprocessor system-on-chips* (MPSoCs) [13] provide the basic architectural framework to design systems which can tackle the processing and power demands of today's applications. MPSoCs integrate multiple, usually heterogeneous, processing elements, a memory hierarchy, an on-chip communication infrastructure, and I/O components into a single chip. Processing elements (PEs) are general purpose processors, digital signal processors, or specialized hardware components tailored to a specific application-domain, e.g., baseband processors for different wireless standards, or cryptography accelerators. One or more PEs have access to local scratchpad memory or cache, which together form a unit called processing module (PM). PMs need to communicate with other PMs, with off-chip memory, or with I/O peripherals such as Ethernet network interfaces. In our design, shown in Figure 1, a network-on-chip (NoC) facilitates the communication between all the components on the chip. A NoC comprises

several routers, physical links between them, and links to NoC interfaces (NoCIFs). The hardware design ensures that no other recipient can access sent messages. Authentication or encryption techniques are needless at the NoC level. A NoCIF provides a component with means to send and receive packets to and from other components. Along with its payload, a packet contains metadata such as receiver and sender address, which allows the routers to establish a dedicated channel directly between two participants.

The described MPSoC paradigm is already used today and allows for a modular system design and an easy integration of different heterogeneous components into one system. However, it does not solve the problem of isolating potentially untrusted components from each other. Every component can communicate with others via the NoC without restrictions. A malicious component could utilize I/O devices, network interfaces, or memory accesses to interfere with other applications or to leak information to other components.

IV. Our Vision: A Trustworthy Hardware/Software Platform Design

Our approach is based on the M^3 microkernel [14] and strives to isolate applications from one another. The isolation is achieved by placing software components on separate hardware "islands". This separation impedes uncontrolled information leakage. Furthermore, isolated applications cannot easily interfere with each other intentionally or unintentionally.

Figure 1 shows our system architecture. Isolation between components is enforced by a system component called *trusted communication unit* (TCU)[1]. It regulates all communication with the NoC and is placed between each NoCIF and its client unit, called *tile*. Typically, a tile hosts a single hardware component, such as a PM, a DRAM interface, or other peripherals. We follow the *isolation-by-default* approach: No communication channels exist unless explicitly allowed and set up by the M^3 microkernel. This kernel runs on a dedicated tile with a general-purpose processor and it is the only entity that is privileged to configure communication channels between the TCUs of all other tiles.

Furthermore, the TCU implements a set of commonly used communication primitives such as message passing and remote direct memory access (RDMA). Source and destination address for such primitives are set by the kernel and strictly enforced by the TCU. If permitted by the kernel, an application can span over multiple different processors, accelerators, and interfaces without putting unrelated applications running on other tiles at risk. Applications form isolated islands of different processing resources. The TCU interfaces with the local NoCIF, the memory, and the PEs of its PM. The NoCIF provides access to the NoC which allows communication with other system components on the chip. The kernel sets up and releases communication channels with other components via the NoCIF by configuring the TCU's internal registers.

[1]In M^3 [14], this component was originally called data transfer unit (DTU). We changed the name to better reflect its purpose.

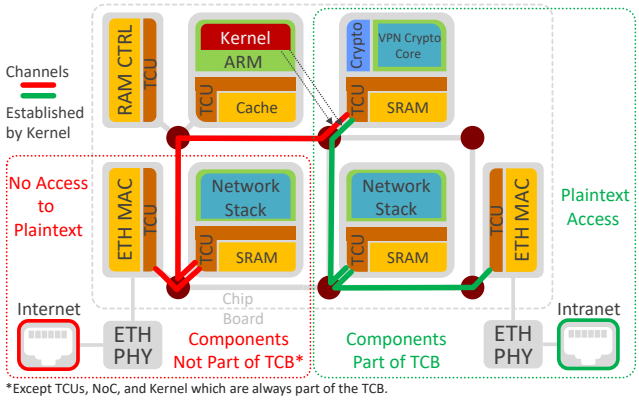

Fig. 2. VPN example implementation on our architecture

The TCU ensures that no data can be sent or received that is not associated with an active channel. A flow-control mechanism prevents applications from performing denial-of-service attacks. The TCU uses the local memory interface to read or write data that is associated with a specific channel within the appropriate local memory address window. This enables the TCU to provide direct memory access (DMA) features.

From a hardware perspective, the TCB of a conventional system contains all components of the MPSoC (IP blocks, processors, accelerators, NoC, and interfaces). If any component is compromised, the security of the system cannot be guaranteed. In contrast, the minimal TCB of our architecture contains only the processor running the OS kernel, the TCU, and the NoC. Besides this small TCB, each application running on the proposed platform has to trust all additional components it uses, for example, a general-purpose processor, a hardware accelerator, and an interface component. Usually, applications only use a certain subset of components within the MPSoC. Hence, the TCB remains as small as possible.

Let us follow up on the VPN example introduced previously. The microkernel approach enables to separate the Internet-facing network adapter, the intranet-facing network interface, and the cryptography layer into separate logical islands. As depicted in Figure 2, each software component is also placed to dedicated physical tiles which are strictly isolated by the TCU. The Internet-facing Ethernet PHY has an associated on-chip I/O peripheral tile (Ethernet MAC), and is isolated by a TCU from other chip components. The kernel establishes a channel from this TCU to the TCU at the tile which runs the Internet-facing network stack. The intranet-facing side is handled in the same way. Now, both tiles running the network stack obtain a dedicated communication channel to the tile which processes the core cryptography algorithms of the VPN service. Every component is now isolated from each other and can communicate only within the bounds imposed by the kernel. Furthermore, attacks like MDS are effectively prevented because trusted and untrusted software no longer share the same processing hardware.

In summary, high-performance, low-latency, energy-efficient, reliable, and secure processing platforms are key to meet the demands of future smart systems. Physical power and latency limitations demand for tightly integrated platforms. Design complexity, cost, and time-to-market constraints require the use of potentially untrusted third-party components. In addition, multiple users and their applications share the same hardware leading to a large TCB. In our approach, the TCU enforces the security policies which are set by the M^3 microkernel. The design of the NoC guarantees that packets are solely routed to the appropriate recipient to eliminate man-in-the-middle attacks. Security-critical software components like the kernel run on dedicated and physically separated tiles to prevent an attacker from compromising the whole system by exploiting software vulnerabilities. Our approach allows applications to only rely on a minimal set of trusted components.

V. Related Work

Building systems in the face of untrustworthy hardware and software components is a key feature of our approach. Strong isolation between processing cores is a prerequisite we share with other works. For example, Intel SGX [15] or Arm TrustZone [16] are commercially deployed solutions for trusted execution environments, so-called enclaves, in general-purpose processors. Enclaves create environments for code which runs in a secure domain isolated from other applications running on the same core. However, the concept fell victim to successful attacks [17] due to their implementation complexity. As a result, low-complexity hardware solutions have been developed. For example, Sancus [18] offers enclave isolation for embedded processors without MMUs or privilege levels. Sanctum [19] extends RISC-V CPUs with minor hardware changes to protect enclaves against cache timing attacks. Nevertheless, CPU side channels were uncovered exploiting caches [20] and other shared states of the CPU [12, 21, 22]. Mitigations with large performance impact [23] were proposed, but strong isolation of processes sharing a core remains challenging. Our tiled architecture can treat entire cores as untrusted building blocks, i.e. they are excluded from the TCB since they are isolated by TCUs. Hence, only exploiting attacks via shared micro-architectural states within CPUs is not sufficient to compromise the whole system.

For compute devices in shared-memory systems, there exist concepts to strengthen the memory isolation. Examples are the standardized IOMMUs [24], and other research work like NoC-MPU [25] or the data protection unit (DPU) [26]. Similarly, SiFive Shield [27] implements an open security architecture for multicore systems with application and memory isolation. The solution also adds cryptographic and entropy-based features to improve the performance of crypto algorithms. Our proposed design unites all these access control enforcement technologies with TCUs which use a hardware-implemented capability system handling both memory accesses (DMA) and communication endpoints (message passing). More importantly, in contrast to these single hardware solutions, our approach improves security by using a hardware/software co-design, i.e. the operating system manages permissions while TCUs enforce the isolation in hardware.

VI. Open Topics

After proposing our solution to improve component isolation using both operating system and hardware innovations, we now turn to open research issues that need to be addressed to round off our vision. We will discuss three topics here: updates, attestation, and supply chain security. However, we are convinced that there are many more research avenues available.

Updates have become a necessity to rejuvenate platform security once a vulnerability has been found. A reactive and long-term update strategy is needed when devices are in use for many years. However, these requirements do not often coincide with market pressure. Our component-based approach will help here because it reduces the necessary amount of maintained software to the set of trusted components. Moreover, many of these trusted components are not use-case specific and will therefore have a larger user base reducing the individual maintenance costs and in general increasing the likelihood of long-term support.

Cooperating software components need to trust each other for certain security goals. When we design our hardware platform as a distributed system of multiple isolated processors, these trust relationships must span across devices. *Attestation* techniques can securely identify remote components using hardware-based trust anchors and cryptographic protocols. An interesting research question is how we can minimize the hardware requirements using our composable hardware/software co-design approach.

Hardware designers nowadays rely on a long and complex design flow and the trustworthiness of the *supply chain*. Once the system is specified using a hardware description language, it is transformed by synthesis and place-and-route into a data format which is then passed to the semiconductor manufacturer. In each of these steps, tools from different vendors are used, which are typically black boxes to the user. How can the designer be sure that no malicious code is inserted into the final design? Moreover, once the design arrives at the manufacturer, lithography masks are produced by yet another supplier. Theoretically at both stages, a well-funded malicious actor could insert hardware trojans or compromise the components in any other, which we rely on in our architecture. Further research is necessary to determine how those issues can be mitigated.

VII. Conclusion

The isolation-by-default approach that is central to our architecture encourages secure-by-design systems at both the hardware and the software level. Our approach combines a componentized, microkernel-based OS with a tile-based hardware architecture and chip-level communication control to construct devices for tomorrow's smart systems. A component-based system design with strong isolation improves both

security and reusability. Vendors can tailor a use-case specific scenario from hardware and software building blocks and reason about the trust relationships between them. The strong isolation primitives from the microkernel at the software level and the TCU at the hardware level ensure that only explicitly allowed communication is possible. Running critical code on low-complexity processor cores designed for security reduces the TCB while untrusted code can benefit from a complex but high-performance processor. This combination reduces the system's overall attack surface.

ACKNOWLEDGMENT

This research is co-financed by public funding of the state of Saxony/Germany.

REFERENCES

[1] M. Condoluci and T. Mahmoodi, "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges," *Computer Networks*, vol. 146, pp. 65–84, 2018.

[2] Cisco Systems, "White paper: Reimagining the End-to-End Mobile Network in the 5G Era," Department of Computer Science, Michigan State University, East Lansing, Michigan, Tech. Rep., 2019.

[3] R. J. Ellison and C. Woody, "Supply-Chain Risk Management: Incorporating Security into Software Development," in *43rd Hawaii International Conference on System Sciences*, Jan 2010, pp. 1–10.

[4] R. Mcilroy, J. Sevcik, T. Tebbi, B. L. Titzer, and T. Verwaest, "Spectre is here to stay: An analysis of side-channels and speculative execution," February 2019. [Online]. Available: https://arxiv.org/abs/1902.05178v1

[5] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter, "The Performance of μKernel-Based Systems." in *SOSP*, vol. 97, no. 29, 1997, pp. 66–77.

[6] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter, "The Nizza Secure-System Architecture," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2005.

[7] O. Arnold, E. Matus, B. Noethen, M. Winter, T. Limberg, and G. Fettweis, "Tomahawk: Parallelism and Heterogeneity in Communications Signal Processing MPSoCs," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, pp. 107:1–107:24, 2014.

[8] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," *CoRR*, vol. abs/1801.01203, 2018.

[9] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)*, Aug. 2018, pp. 973–990.

[10] K. N. Khasawneh, E. M. Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "SafeSpec: Banishing the Spectre of a Meltdown with Leakage-Free Speculation," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[11] M. Schwarz, M. Lipp, C. Canella, R. Schilling, F. Kargl, and D. Gruss, "ConTExT: A Generic Approach for Mitigating Spectre," in *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[12] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout: Leaking Data on Meltdown-Resistant CPUs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, 2019, p. 769–784.

[13] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor System-on-Chip (MPSoC) Technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, Oct 2008.

[14] N. Asmussen, M. Völp, B. Nöthen, H. Härtig, and G. Fettweis, "M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2016, pp. 189–203.

[15] "Intel Software Guard Extensions (Intel SGX)," 2021. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html

[16] "Trustzone for Cortex A – TEE Reference Documentation," 2021. [Online]. Available: https://www.arm.com/why-arm/technologies/trustzone-for-cortex-a/tee-reference-documentation

[17] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on Intel SGX," *arXiv preprint arXiv:2006.13598*, 2020.

[18] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 479–498.

[19] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874.

[20] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, and W. Kunz, "Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 994–999.

[21] J. Stecklina and T. Prescher, "Lazyfp: Leaking FPU Register State using Microarchitectural Side-Channels," *arXiv preprint arXiv:1806.07480*, 2018.

[22] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad: Cross-privilege-boundary data sampling," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 753–768.

[23] A. Prout, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, P. Michaleas, L. Milechin, J. Mullen, A. Rosa, S. Samsi, C. Yee, A. Reuther, and J. Kepner, "Measuring the Impact of Spectre and Meltdown," in *2018 IEEE High Performance Extreme Computing Conference, HPEC 2018, Waltham, MA, USA, September 25-27, 2018*. IEEE, 2018, pp. 1–5.

[24] Advanced Micro Devices Inc., "AMD I/O Virtualization Technology (IOMMU) Specification," 2021. [Online]. Available: https://www.amd.com/system/files/TechDocs/48882_IOMMU.pdf

[25] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE'11, March 2011, pp. 1–4.

[26] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure Memory Accesses on Networks-on-Chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1216–1229, Sept 2008.

[27] J. Prior, "SiFive Shield: An Open, Scalable Platform Architecture for Security," 2019. [Online]. Available: https://www.sifive.com/blog/sifive-shield-an-open-scalable-platform-architecture