Trustworthy Computing for O-RAN: Security in a Latency-Sensitive Environment

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

***Abstract*—New 5G/6G mobile networks will allow to run distributed multi-tenancy workloads with high requirements on latency, throughput, and energy efficiency. This also demands a transformation of the underlying radio access networks (RANs) towards an architecture with open-source software, heterogeneous hardware, and interoperable interfaces as already described by the O-RAN ALLIANCE. In principle, virtualization of the RAN allows for defining additional interfaces which can be used for system verification and therefore can increase the trustworthiness of implementations. However, due to many security risks induced by the current O-RAN specification, it requires to rethink the overall security architecture from the ground up to establish trustworthy mobile networks in general.**

In this paper, we discuss hardware-enforced capabilities as a structuring principle for future O-RAN architectures. We present our approach of a hardware/operating system co-design to implement these principles. The evaluation results of architectural components demonstrate the technical feasibility of our approach and illustrate that security and low latency can be achieved simultaneously.

***Index Terms*—O-RAN, Operating System, Tiled Architecture, Security, Latency**

I. Introduction

Our digital world is driven by highly distributed systems. Current (5G) and future (6G) mobile networks are prime examples. Such systems consist of a multitude of heterogeneous components – and not all of them can be considered trustworthy. Upcoming radio access network (RAN) architectures like O-RAN as specified by the O-RAN ALLIANCE [1] start to address this issue by making the RAN more independent from proprietary components. This means to define open protocols and interfaces as well as to abstract network elements and functions. The goal is a modularized design to establish a multi-vendor, interoperable, and trustworthy architecture. In fact, the integration of possibly untrusted components is not limited to software but will extend to hardware components, e.g., AI accelerators, necessary to power the highly AI-driven O-RAN architecture. Bringing computation into the network and to the edge, a 6G network has to run distributed multi-tenancy workloads with high requirements on latency, throughput, and energy efficiency.

At the same time, future networks must meet the highest security and safety requirements, since they are part of the critical infrastructure supporting many use cases from the domain of cyber-physical systems like the Internet of Things. As these systems can directly influence the physical world, there is a high risk of directly harming people in case of malfunction. Therefore, future networks must embed effective security controls to implement the required protection goals such as confidentiality, integrity, and availability. They also have to be resilient and preserve the privacy of their users [2].

Designing a trustworthy system which balances the requirements from the domains of utility as well as security and safety in a meaningful way is a challenging task. Energy efficiency and low latency require close interaction of the heterogeneous components, whereas safety and security demand for strong isolation of them. The necessary security controls often induce costs in terms of increased latency or higher power consumption due to cryptographic operations.

Although the current specifications of O-RAN induce many security risks [3], the openness of the overall architecture and the ongoing work with respect to security – e.g. within the O-RAN Security Focus Group (SFG) and the related next Generation Research Groups – allow to rethink the overall security architecture from the ground up. In particular, this includes a suitable hardware architecture that supports a secure integration of potentially untrusted components while maintaining energy efficiency, low latency, and scalability.

The main contributions of this paper are:

- We discuss *hardware-enforced capabilities* as a structuring principle for a future O-RAN architecture. We claim that capabilities unify the design goals of security and low latency.
- We present our approach of a *hardware/operating-system co-design* to implement these principles. The resulting platform combines the performance of hardware-rooted security primitives with the upgradeability and flexibility of software-implemented policy decisions.
- We point out existing evaluation results for architectural components to demonstrate the technical feasibility of our architecture. We illustrate that security and low latency can be achieved simultaneously.

In the next section, we explain our notion of trustworthiness, the guiding principles for its implementation, and derive open research questions. Section III details our approach for a secure hardware/operating system co-design, for which Section IV presents a proof-of-concept evaluation. Afterwards, we discuss related work and conclude.

978-1-6654-5975-4/22/$31.00 © 2022 IEEE

II. Trustworthiness on the Hardware Interface

A key requirement for O-RAN deployments is their ability to serve both the primary functionality of the mobile network as well as host applications from third-party vendors. Hence, the O-RAN infrastructure now shares its internal resources, e.g., compute resources (CPUs, GPUs, accelerators), memories, and interfaces with third-party code. This resource sharing introduces new obligations to protect against attacks on the software and hardware levels in addition to the existing attack surface at the network-interface level.

A. Attack Surfaces and Resulting System Requirements

These new attack scenarios include software-level attacks on application programming interfaces of the local operating-system (OS), like an application exploiting a vulnerability in the OS kernel to elevate itself to a higher privilege level. Similarly, applications in a virtualized environment can attack the hypervisor to escape from their virtual machine. Attacks can also be mounted on shared hardware resources to extract secrets from co-running applications. The Spectre [4] and Meltdown [5] attacks have proven that vulnerabilities in the underlying processors can nullify any OS protection and enable cross-application attacks by directly exploiting hardware-design problems. In addition, supply-chain attacks can introduce hard-to-detect deliberate backdoors into manufactured hardware components. Current system architectures typically assume that hardware is fully trusted and consequently do not protect against such scenarios.

We therefore define a notion of trustworthiness, which includes secure and reliable operation anchored within the hardware. Upper layers can use these hardware-anchored security primitives to implement their security boundaries. A trustworthy hardware design must not assume that all processors are secure but must tolerate accidentally or maliciously broken compute resources. On the chip level, a tiled hardware architecture [6] allows to integrate resources into different compartments, connected via a network-on-chip. Security is achieved by isolating the hardware tiles from each other, resulting in a distributed system of mutually distrusting participants.

However, these isolated resources need to collaborate in order to work on a common task. Therefore, the system must offer primitives to break the isolation in a controlled way to allow collaboration between tiles. These control mechanisms however must not be at the discretion of the resources we want to isolate but must be enforced upon them from the outside. The system's trusted computing base (TCB) is the set of all components in the system we must rely upon for correct operation. We postulate that it is beneficial for security if this reliance set is of minimal complexity. We therefore propose a system design with a minimal TCB, which includes only the hardware mechanism for communication control and the components programming this mechanism. The resulting system can securely integrate untrusted third-party hardware components and third-party software applications, because the TCB enforces isolation and controlled communication for these components. The enforcement is situated at the hardware level, thus protecting against hardware-level attacks and providing the necessary foundation for higher-level protection at the software level. By protecting against malicious behavior, the system also protects against accidental misbehavior, so the same architecture principle also improves the structural reliability and safety of the system.

Two questions remain to be discussed: How can such a minimal communication control mechanism look like and how can it be programmed? Considering that an O-RAN system needs to dynamically adapt to changing load requirements and application configurations, our system must be able to dynamically reprogram the hardware communication control from software. This requirement raises the question of how the interface between hardware and software should be designed.

B. Capabilities as a Common Hardware Interface

Today, isolation mechanisms in hardware/software systems are often based on virtualization, with communication control being implemented by network-level routing. This is a disadvantage because they are very complex mechanisms. The TCB contains the complete processors because they isolate different virtual machines from each other, the network interface cards because they enforce communication control, and the hypervisor software layer for programming everything. The interfaces between these components are unnecessarily broad, resulting in such a complex TCB.

We propose to rethink this interface and offer *hardware-enforced capabilities* as its central primitive. A *capability* is a secure reference to a resource. In a *capability system*, all resource accesses are governed by capabilities. Participants in such a system own a set of capabilities, which has been given to them. They are unable to forge capabilities or to create them from thin air. When communicating with another entity in the system, communication is only allowed when the sender possesses a capability to communicate with that entity.

We propose to implement capabilities in hardware: all communication initiated by a compute resource on one tile and directed to another tile is policed by an independent *guard component*, which always checks the sender for the presence of a valid capability to the target tile. This capability-based communication control has three advantages:

Deny by default: Without a valid capability, no communication is allowed. Thus, a system initially offers complete isolation, similar to a firewall based on an allow list.

Resource agnostic: Because the capability checks are implemented at the hardware level external to the resource being controlled, the mechanism is independent of the resource. The same mechanism can be used for application software running on a general-purpose processor but also for programmable accelerators like GPUs as well as fixed-function accelerators not running any software at all.

Pass by reference: Capabilities can be passed as payload when communicating with another entity in the system. As the capability is a resource reference, this transfer can securely pass access to a resource between communication partners

without transferring the resource itself. This can help to reduce unnecessary data copies by passing a capability to the data between partners, without compromising security.

These benefits are a perfect fit for our design goals of security and efficiency. Security is achieved through a small-TCB design and the deny-by-default behavior of capabilities. Efficiency is enabled by the transparent use of energy-saving accelerators for specific workloads and the elision of unnecessary data copies through the pass-by-reference property of capabilities. However, it remains to be demonstrated, that permanent checking of capabilities on heavily used communication paths does not unduly impede communication latency. Also, the right separation of concerns must be explored between capability enforcement in hardware, which is fast but immutable, and the programming of communication control from software, which is slow but upgradeable and flexible.

III. Our Approach: Secure Hardware/Operating System Platform

To address the aforementioned requirements of O-RAN architectures, we propose a scalable system platform that provides the necessary security features including hardware-enforced capabilities. The system is organized hierarchically with comparable building blocks at each level of the hierarchy (see Figure 1). At each level, different resources are isolated by guard components to selectively allow communication. Besides these similarities between layers, end-to-end latencies for data processing and communication scale from microseconds on tile-level, milliseconds between tiles and chips, towards multiple seconds in a distributed system. This induces the need to investigate different approaches to interface hardware and software, e.g., to check capabilities on the communication path, or to efficiently distribute the workload of the applications.

A. M³: A Secure Tiled Chip Architecture

On tile- and chip-level, the compute resources are made up of processors and application-specific accelerators. Typically, both are third-party hardware blocks and should therefore not be trusted. For this hierarchy level, we propose the M³ system [7], an integrated hardware/OS platform that supports the secure integration of untrusted components. M³ is based on a tiled hardware architecture and a network-on-chip (NoC) for communication between the tiles (see Figure 1b). As the guard component, M³ introduced a new hardware component into each tile called *trusted communication unit* (TCU)[1]. Each TCU contains a set of capabilities, which is initially empty, and defines which tile-external resources a tile can access. Therefore, tiles are isolated by default, but if the corresponding capability is available, the TCU can be used to perform DMA-like memory transfers and message passing.

On the software side, M³ provides a microkernel-based OS. The kernel runs on a dedicated tile with a general-purpose processor and is the only master which is responsible to configure the capability sets in the TCUs of the remaining tiles. The M³ kernel also supports exchanging capabilities, allowing to pass data by reference between applications. As each tile contains a TCU, the TCUs provide a uniform interface for the heterogeneous tiles with potentially different general-purpose cores, accelerators, and memory. This simplifies both the management of such an heterogeneous system for the M³ kernel and also the usage of the system by applications.

The proposed TCB in M³ consists of the NoC, the processor that runs the M³ kernel, and the TCUs. The NoC must be part of the TCB to ensure that packets reach their destination without being manipulated. Since the kernel can configure the capability sets in all TCUs, the kernel including its underlying processor also must be trusted. The TCUs need to be trusted as well to ensure that tiles are isolated from each other and access to tile-external resources is only granted as permitted by the capability set. The trust into the TCU can be strengthened by formal verification.

B. FractOS: A Distributed Operating System

FractOS [8] takes many of the M³ concepts and lifts them from chip-level to distributed-system-level. Instead of tiles being separated by a NoC, we are looking at individual machines separated by a data-center network. The machines can house a general-purpose processor, but they can also contain other resources, for example, accelerators like GPUs or TPUs/NPUs, but also DRAM or SSD storage (see Figure 1a). Thus, FractOS features similar support for system heterogeneity as M³.

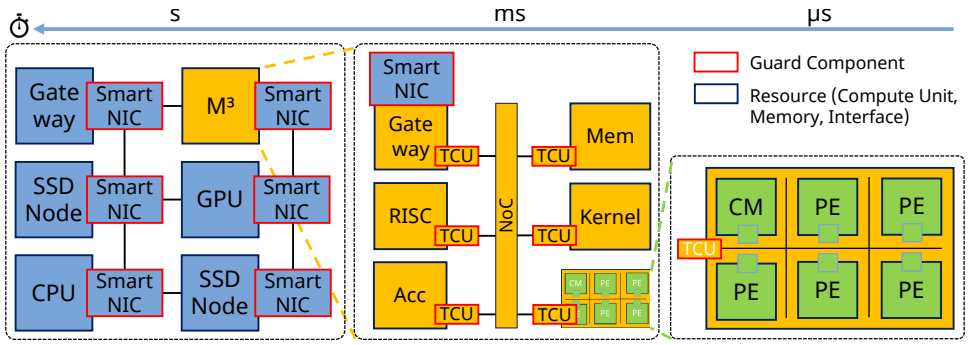
The role of the guard component is implemented by SmartNICs, which are placed in front of every compute resource. A SmartNIC is a network interface card (NIC), which can run software independently from the connected machines to inspect the passing network traffic. Like the NoC in M³, the data-center infrastructure in terms of cables and switches must be trusted, but the hardware and software connected to it remains untrusted, because the SmartNIC acts as a trusted guard component. Due to different message delivery time scales on a data-center network compared to a NoC, the capability checks are implemented on the SmartNIC in software.

FractOS uses a capability system similar to M³ but does not employ a single or a few instances of a kernel component as this would quickly become a bottleneck in large-scale deployments. Instead, the kernel's responsibilities are fully distributed across all participating SmartNICs, each of them independently managing capabilities for the resources connected to it. Capability invocation and delegation, which are critical operations, do not require any interaction with centralized services. With these implementation choices, FractOS allows secure capability-based communication with low latency and high scalability in a distributed data-center-scale setup.

C. Trading Security Against Latency

The two presented hardware/OS platforms, M³ and FractOS, enable secure communication based on capabilities, either checked in hardware or in software. However, some use

[1]In M³ [7], this component was originally called data transfer unit (DTU). We changed the name to better reflect its security properties.



(a) FractOS (b) M³ (c) App-specific tile within M³

Fig. 1: Our approach on a secure hardware/OS platform. Multiple resources are organized hierarchically. Guard components enforce security policies and gateways enable communication across layers. FractOS may enclose one or more M³ systems, which in turn may enclose application-specific tiles with a CoreManager (CM) for low-latency data processing.

cases in 5G/6G networks even require end-to-end latencies of below 1 ms [6]. This means that the latency of the actual baseband processing and the data flow between the chip's compute resources should be in the range of microseconds or even nanoseconds. For this purpose, capability checks can be skipped to reduce the overall processing delay to a minimum. For example, we could add another level within the M³ hierarchy by creating an application-specific tile (for short *app-tile*). As depicted in Figure 1c, this app-tile may include multiple processing elements (PEs) with application-specific accelerators (e.g., DSPs) and its own on-chip network. While the TCU of this app-tile still checks its capabilities, all components within the tile are not isolated by TCUs. This allows to achieve even lower latencies compared to the standard M³ approach. Due to the lack of isolation within the app-tile, we trade security against latency.

As a consequence, the hard- and software components in the app-tile are not managed by the M³ kernel anymore. We think it requires a dedicated scheduling unit, which we call *CoreManager* (CM), to distribute application-specific workloads within the app-tile. The CM may be implemented as a simple hardware component that applies a given data-flow graph configured by the M³ OS. Hence, latency is reduced again: while a distinct workload is processed, any dynamic configuration to set up communication is directly forwarded to the CM without involving the OS.

IV. Evaluation

In the evaluation, we strive to answer the questions of whether security and low latency are a contradiction and how beneficial the ability to pass data by reference is. We start with a study of M³, followed by an evaluation of FractOS.

A. Communication Latency vs. Isolation with M³

The M³ hardware/software platform targets the chip level in the described hierarchy and uses TCUs as guard components to control each tile's communication.To evaluate the costs of this additional isolation, we perform a comparison with Linux, a widespread and well-optimized OS. As M³ is a hardware/OS co-design, we use a custom FPGA platform to perform the comparison. The FPGA platform consists of a tiled architecture with multiple compute tiles, each containing a RISC-V BOOM core and a TCU. The BOOM core is clocked with 80 MHz to fully meet timing requirements during FPGA synthesis and place-and-route.

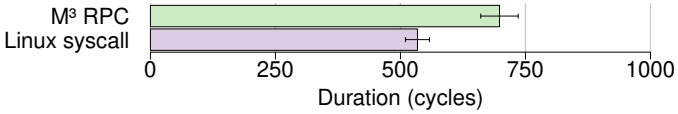


Fig. 2: Latency of TCU-based communication on M³ and a Linux system call as a reference.

Applications on M³ use remote procedure calls (RPCs), consisting of a request and a response, to access system services or the M³ kernel. Therefore, we measured the performance of no-op RPCs (i.e., only showing the costs of the RPC itself) between tiles on M³ and show the performance of no-op system calls on Linux as a reference. Both M³ and Linux run on our FPGA platform. M³ runs the communication partners on two BOOM cores, whereas Linux uses a single BOOM core. We performed 1000 runs with a warm system.

The results in Figure 2 show that cross-tile communication is with 698 cycles only slightly slower than a system call on Linux with 534 cycles. Note that 698 cycles translate into 8.7 μs with the 80 MHz core but would translate into 0.7 μs with a more realistic clock of 1 GHz. On M³, the RPC requires multiple interactions with the TCU to send the message, fetch the message on the receiver side, reply on the received message, fetch the reply on the sender side, and mark the message as read. For all these TCU interactions, the TCU verifies their validity based on the available capabilities.

B. Communication Latency vs. Isolation with FractOS

FractOS is designed for the distributed-system level and can be deployed within a data center. The guard component is implemented based on SmartNICs, with one SmartNIC per node to control the resource access of every node. Experiments are performed on a 3-node cluster with the characteristics listed in Table 1.

Similar to the experiment with M³, we evaluate the latency of invoking a no-op syscall in FractOS. The results shown in

Host CPU	Intel Xeon E5-2620 v2
Host memory	64 GB, DDR3 @ 1333 MHz
SSD	Samsung 970evo Plus (500GB)
GPU	NVIDIA Tesla K80
Smart NIC	Mellanox BlueField MT416842, RoCEv2
Network	10 Gbps fabric and switch (split cables)

Table 1: FractOS evaluation environment.

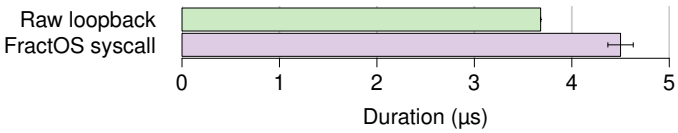


Fig. 3: Latency of a no-op FractOS syscall, compared to raw loopback latency.

Figure 3 compare the raw loopback latency (measured with the `ibv_rc_pingpong` benchmark) with the latency for FractOS syscalls. As with M^3, FractOS adds some overhead to verify the validity of the operations based on capabilities, which results in the additional 0.8 μs latency for FractOS. Note that these measurements show the latency of a syscall within a single node. RPC operations between different nodes in the same data center or across data centers can easily result in latencies in the millisecond range.

C. Reducing Data Transfers with FractOS

Finally, we evaluate the benefit of being able to pass data around by reference via capabilities. To that end, we use the storage stack and GPU service of FractOS in a face-verification service and compare it to the conventional solution in disaggregated data centers. The face-verification service verifies the identity of a person by matching the photo and the ID in the input with the photo corresponding to that ID from a database. The database is stored on a storage device and the match algorithm runs on the GPU. When receiving a face-verification request from a client, the benchmark builds a pipeline of requests to (1) open and read the files from storage into the GPU, (2) execute the face-verification GPU kernel, (3) copy the results from the GPU into the application memory, and (4) send a response to the client.

As the conventional solution, we use a frontend node that fetches files from a remote ext4 file system via NFS. The file system is backed by NVMe-over-Fabrics storage. Both NFS and NVMe-oF use in-kernel drivers in Linux. The image data is then copied to and processed by a remote GPU via rCUDA. On FractOS, both the GPU service and the storage stack use capabilities as their interface. Namely, the GPU service expects a capability to the memory the GPU should compute on and an RPC capability to inform another party about the completion of the execution. The storage stack of FractOS consists of a simple file system and an NVMe driver for the storage device. In contrast to conventional file systems, FractOS' file system provides the calling application with an RPC capability directly to the NVMe driver. In other words, the application can bypass the file system during data access and directly request the data from the NVMe driver. FractOS' capability system and the involved services make sure that the direct access to the NVMe driver is secure.

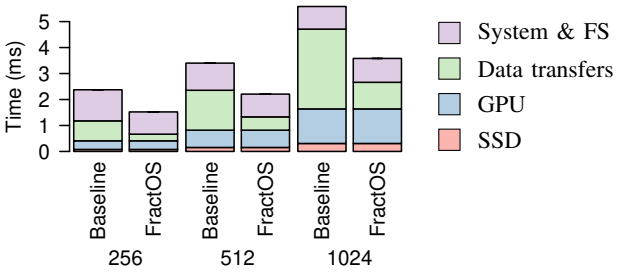


Fig. 4: End-to-end latency of a face verification request for different image batch sizes (256, 512, and 1024 images).

Figure 4 shows the latency for different image batch sizes of a single client. In the baseline, data is transferred over network three times: over NVMe-oF, NFS, and rCUDA. FractOS can optimize the data path down to a single transfer: from NVMe directly to GPU. This benefit manifests in lower per-request latencies for FractOS. Additionally, FractOS reduces the number of messages on the control path. For the conventional solution, all requests travel from the frontend to one component (e.g., NVMe) and back to the frontend, resulting in a total of eight control messages (two for open, four for reading, two for GPU). FractOS reduces this to five messages.

In summary, capabilities have a small overhead during communication but allow to reduce the number of control messages and to pass data by reference in a secure manner. This reduction can even result in significant speedups as shown with FractOS. Note that M^3 achieves the same benefits via its capability system as FractOS, but we do not show it in the evaluation for brevity.

V. Related Work

Security in O-RAN architectures: Since O-RAN is a comparatively new concept which is still under development, not many solutions addressing the security issues can be found in the literature. In fact, papers and reports are mainly concerned with the emerging security risks [9]. The O-RAN ALLIANCE has established a working group focusing on security. Although their work has led to many security improvements, the currently published security specifications are still lacking a comprehensive solution, as recently revealed by a study [3]. The proposed security measures [10] especially assume a trustworthy execution environment (covering hardware, virtualization platform and the operation of it) and do not adequately address the supply chain risks nor the possibility of compromised components (insider attacks) we address.

Isolation of applications and compute resources: Strong isolation between applications is a prerequisite to host multiple applications from third-party vendors on a shared O-RAN infrastructure. For that purpose, commercially available solutions in CPUs such as Intel SGX, TDX, or Arm TrustZone are deployed to create *trusted execution environments* (TEEs). The code in a TEE runs in a secure domain isolated from other applications running on the same processor. However, the

concept fell victim to successful attacks [4, 5] due to its implementation complexity. As a result, low-complexity hardware solutions have been developed, for example, to protect TEEs against cache-timing attacks [11]. Nevertheless, side-channel attacks are still discovered on a regular basis, which exploit caches and other shared states of the CPU [12]. Mitigations are typically complex as well and lead to performance losses, hence strong isolation of processes sharing a core remains challenging. Our proposed concept on a hardware/OS platform can treat entire compute resources as untrusted building blocks, i.e. they are excluded from the TCB because they are isolated by guard components. Since different compute resources do not share micro-architectural states, these attacks cannot be mounted on other compute resources.

Capability systems: Capability systems have already been introduced some time ago [13]. Similarly to M^3 and FractOS, other works utilize or design a specific capability system to manage communication permissions between resources. For example in SemperOS [14], capabilities are designed and adapted to a microkernel-based OS in a distributed setting. CapNet [15] is a capability-based architecture for large-scale distributed systems like a cloud network. Barrelfish [16] also employs a capability system and spreads it over a distributed system of processes. In contrast to FractOS, the mentioned approaches disregard reliability, which is a drawback when scaling the system over a network. Furthermore, our hardware/OS platform concept considers enforcing capabilities in a separate hardware guard component to place trustworthiness into the system.

VI. Conclusion and Future Work

We presented our approach for a hardware/operating system co-design to implement hardware-enforced capabilities as a structuring principle for future O-RAN architectures. On tile- and chip-level, we proposed the M^3 system that enables the secure integration of untrusted hard- and software components. This modularized design may allow to build a multi-vendor and trustworthy architecture as required by the O-RAN specification. Furthermore, the components of M^3 are released as open source[2] to support the O-RAN's idea of an open ecosystem. FractOS takes many of the M^3 concepts and applies them on a distributed-system level in data centers. Measurement results of both approaches show that capabilities lead to a small latency overhead during communication but allow to reduce the number of control messages and to pass data by reference in a secure manner. In M^3, capability checks are implemented in hardware by TCUs to benefit from a deterministic and low-latency communication control as well as to meet the tight limitations on energy consumption and chip size. In contrast, FractOS performs the capability checks in software on the SmartNIC. The flexibility of a software implementation is essential, e.g., to handle node crashes, which are common in data centers with many nodes.

[2]https://github.com/Barkhausen-Institut

We demonstrated the technical feasibility of our approaches by only using architectural components of O-RAN infrastructures. It remains future work on how to implement a fully trustworthy and low-latency hardware/software platform that provides the foundation for O-RAN systems. In particular, the interfaces between FractOS and M^3 as well as between the M^3 kernel and CoreManager have to be investigated.

Acknowledgment

This research is financed on the basis of the budget passed by the Saxon State Parliament in Germany.

References

[1] "O-RAN specifications," O-RAN ALLIANCE. [Online]. Available: https://www.o-ran.org/specifications

[2] G. P. Fettweis and H. Boche, "On 6G and Trustworthiness," *Communications of the ACM*, vol. 65, no. 4, pp. 48–49, 2022.

[3] S. Köpsell, A. Ruzhanskiy, A. Hecker, D. Stachorra, and N. Franchi, "Open RAN Risk Analysis," Federal Office for Information Security, BSI studies, February 2022. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/5G/5GRAN-Risk-Analysis.html

[4] 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," *meltdownattack.com*, 2018. [Online]. Available: https://spectreattack.com/spectre.pdf

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *meltdownattack.com*, 2018. [Online]. Available: https://meltdownattack.com/meltdown.pdf

[6] G. Fettweis, M. Hassler, R. Wittig, E. Matus, S. Damjancevic, S. Haas, F. Pauls, S. Nam, and N. Grigoryan, "A Low-Power Scalable Signal Processing Chip Platform for 5G and Beyond - Kachel," in *53rd Asilomar Conference on Signals, Systems, and Computers*, 2019.

[7] N. Asmussen, S. Haas, C. Weinhold, T. Miemietz, and M. Roitzsch, "Efficient and Scalable Core Multiplexing with M^{3}v," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Feb. 2022, p. 452–466.

[8] L. Vilanova, L. Maudlej, S. Bergman, T. Miemietz, M. Hille, N. Asmussen, M. Roitzsch, H. Härtig, and M. Silberstein, "Slashing the Disaggregation Tax in Heterogeneous Data Centers with FractOS," in *European Conference on Computer Systems (EuroSys)*, Rennes, France, April 2022, p. 352–367.

[9] O-RAN ALLIANCE e.V., "O-RAN Security Threat Modeling and Remediation Analysis," Technical Specification, O-RAN SFG: Security Focus Group, O-RAN.SFG.Threat-Model-v03.00, April 2022.

[10] ——, "O-RAN Security Requirements Specifications," Technical Specification, O-RAN SFG: Security Focus Group, O-RAN.SFG.Security-Requirements-Specifications-v03.00, March 2022.

[11] 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.

[12] C. Canella *et al.*, "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] J. S. Shapiro, J. M. Smith, and D. J. Farber, "EROS: a fast capability system," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999, pp. 170–185.

[14] M. Hille, N. Asmussen, P. Bhatotia, and H. Härtig, "SemperOS: A Distributed Capability System," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 709–722.

[15] A. Burtsev, D. Johnson, J. Kunz, E. Eide, and J. E. van der Merwe, "CapNet: Security and Least Authority in a Capability-Enabled Cloud," in *Proceedings of the 2017 Symposium on Cloud Computing (SoCC 2017), Santa Clara, CA, USA*, September 2017, pp. 128–141.

[16] A. Baumann, P. Barham, P. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: a new OS architecture for scalable multicore systems," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, 2009, pp. 29–44.