# HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment

Pascal Nasahl
pascal.nasahl@iaik.tugraz.at
Graz University of Technology

Robert Schilling
robert.schilling@iaik.tugraz.at
Graz University of Technology

Mario Werner
mario.werner@iaik.tugraz.at
Graz University of Technology

Stefan Mangard
stefan.mangard@iaik.tugraz.at
Graz University of Technology
Lamarr Security Research

## ABSTRACT

To ensure secure and trustworthy execution of applications in potentially insecure environments, vendors frequently embed trusted execution environments (TEE) into their systems. Applications executed in this safe, isolated space are protected from adversaries, including a malicious operating system. TEEs are usually build by integrating protection mechanisms directly into the processor or by using dedicated external secure elements. However, both of these approaches only cover a narrow threat model resulting in limited security guarantees. Enclaves nested into the application processor typically provide weak isolation between the secure and non-secure domain, especially when considering side-channel attacks. Although external secure elements do provide strong isolation, the slow communication interface to the application processor is exposed to adversaries and restricts the use cases. Independently of the used approach, TEEs often lack the possibility to establish secure communication to peripherals, and most operating systems executed inside TEEs do not provide state-of-the-art defense strategies, making them vulnerable to various attacks.

We argue that TEEs, such as Intel SGX or ARM TrustZone, implemented on the main application processor, are insecure, especially when considering side-channel attacks. In this paper, we demonstrate how a heterogeneous multicore architecture can be utilized to realize a secure TEE design. We directly embed a secure processor into our HECTOR-V architecture to provide strong isolation between the secure and non-secure domain. The tight coupling of the TEE and the application processor enables HECTOR-V to provide mechanisms for establishing secure communication channels between different devices. We further introduce RISC-V Secure Co-Processor (RVSCP), a security-hardened processor tailored for TEEs. To secure applications executed inside the TEE, RVSCP provides hardware enforced control-flow integrity and rigorously restricts I/O accesses to certain execution states. RVSCP reduces the trusted computing base to a minimum by providing operating system services directly in hardware.

## KEYWORDS

TEE; secure I/O; heterogeneous computer architecture; RISC-V

## 1 INTRODUCTION

With the growing demand for complex IT applications, such as autonomous driving or smart city infrastructures, software complexity increases steadily. The codebase of Linux, for example, increases by 250k lines of code each year, reaching 27.8M LOC in 2020 [13]. This is challenging because one can expect roughly 1 to 25 bugs in 1,000 lines of code [39]. While not all of these bugs might be exploitable by an attacker, a growing codebase complexity clearly leads to a larger attack surface. One strategy to deal with the growing complexity and meet security goals is to isolate all security critical applications using a trusted execution environment (TEE) [25].

A TEE establishes a secure execution environment by creating a safe, isolated space using hardware and software primitives. Most TEE threat models consider a powerful adversary controlling user space applications, the operating system, or even the hypervisor, trying to influence the execution of applications in the trusted environment. To achieve the protection needed for this threat model, TEEs require strong isolation between the TEE and the rich execution environment (REE).

The powerful concept of shifting security critical applications into a trusted execution environment is already adapted by major vendors like Intel, ARM, and Apple. One of the most common TEE design approaches is to create a virtual secure processor in the main application processor by using hardware extensions. Intel SGX [22] and ARM TrustZone [7] take this approach. Contradictory to embedding the TEE tightly into the CPU, Google's Titan [32] and Apple's T2 [4] implements a secure element by externally mounting a dedicated security processor next to the main CPU. However, both TEE design approaches yield different weaknesses. Several recent attacks [15, 16, 20, 27, 28, 36, 49, 55, 65] showed that the isolation of TrustZone and SGX can be bypassed by mounting cache or transient-based side-channel attacks. While dedicated secure elements provide strong isolation between REE and TEE, Google's Titan, e.g., uses a slow SPI connection for communication between the two domains limiting potential use cases. Furthermore, this off-chip communication fabric between REE and TEE is physically exposed to an attacker, making it vulnerable against probing attacks.

Independently of the used TEE design approach, typical TEE implementations offer several weaknesses. First, although a security breach within the TEE is fatal, operating systems deployed in the secure environment surprisingly do not offer state-of-the-art protection mechanisms like ASLR, guard pages, or stack cookies [17]. Second, most TEEs do not provide architectural features to establish secure communication with I/O devices. The lack of trusted I/O paths in TEE systems is critical because secrets shared between user and TEE are unprotected.

With the Intel Lakefield architecture [33] and an AMD patent [40], major vendors recently announced to introduce heterogeneous multi-core architectures in upcoming processor designs. While the approach of tightly coupling different processor cores on one chip to balance power and energy efficiency is already used by ARM's big.LITTLE technology [6] in mobile platforms, Intel and AMD are planning to introduce this concept in forthcoming computer architectures. This design strategy raises the following research question: *RQ1: "Can the tight coupling of distinct processor cores on a SoC be used to increase the security of trusted execution environments?"*

## Contribution

In this paper, we introduce HECTOR-V, a design methodology utilizing a heterogeneous multi-core architecture to develop secure trusted execution environments. HECTOR-V achieves strong isolation between the secure and non-secure domain and provides protection for various side-channel attacks, such as cache and transient-based attacks, by using a distinct processor of the heterogeneous core cluster for the secure environment. The tight coupling of TEE and REE in the shared SoC infrastructure yields several advantages. In the HECTOR-V architecture, the application processor and TEE are connected through an interconnect, enabling a high-speed link between the two processors. Additionally, since the TEE is embedded into the SoC, all peripherals integrated into the system are also available for the secure environment. To manage peripheral sharing and protect peripherals from unauthorized accesses, HECTOR-V further introduces a secure I/O path concept. The identifier-based strategy deeply nested into the communication fabric and the peripherals allows a fine granular protection of the attached peripherals. In HECTOR-V, the access permission management is implemented using a hardware-based security monitor. While the owner of the security monitor can configure a set of access permissions, the other parties in the system can request access to certain peripherals by consulting the security monitor. We further extend this mechanism to dynamically grant and deny access to certain peripherals by introducing the concept of security monitor ownership transfer. To enable various use cases, the owner of the security monitor dynamically can transfer the ownership to other parties. Furthermore, we introduce RVSCP, a security-hardened RISC-V processor tailored for being used as a TEE. Although the HECTOR-V architecture is independent of the used TEE processor, we show that RVSCP further increases the TEE security by utilizing features of HECTOR-V. RVSCP isolates applications within the trusted execution environment by enforcing the integrity of the control-flow. We combine the control-flow information with the secure I/O mechanism to only grant access to certain peripherals when reaching a predefined control-flow state. To reduce the trusted computing base and therefore the attack surface to a minimum, RVSCP implements operating system features, such as multitasking, directly in hardware. We demonstrate the HECTOR-V concept by introducing a heterogeneous multi-core architecture for RISC-V. We embedded a RISC-V processor with a control-flow integrity unit into the HECTOR-V architecture. To verify the functionality of HECTOR-V and RVSCP design approaches, we use secure boot as a prototype application on our FPGA implementation. In summary, our contributions are:

**HECTOR-V,** a design methodology for developing secure TEEs. HECTOR-V utilizes a heterogeneous multi-core architecture to realize a secure trusted execution environment. The architecture includes a configurable security monitor managing access permissions to specific peripherals. This mechanism allows the system to establish secure communication channels between peripherals, users, and the processor cores. The dynamic transfer of security monitor ownership enables the HECTOR-V architecture to realize several use cases, such as secure boot or executing trusted applications.

**RVSCP,** a security-hardened processor integrated into the HECTOR-V architecture. RVSCP protects applications within the TEE by combining a previously introduced control-flow integrity unit with the peripheral access protection offered by HECTOR-V. The secure processor provides operating system features in hardware to minimize the trusted computing base.

## 2 BACKGROUND

To protect the processing of sensitive data, such as biometric data or cryptographic keys in an untrusted environment, trusted execution environments (TEE) are used. These secure environments comprise a combination of hardware and software features, the trusted computing base (TCB) [25]. A TEE guarantees the secure execution of trusted applications, even when considering a malicious operating system running in the rich execution environment (REE). This section summarizes popular TEE design strategies.

**Virtual Processor based TEEs.** Virtual processor based TEEs extend the main processor with additional hardware and software features to establish a secure, isolated space within this processor. As this approach utilizes the main processor for the TEE, such TEEs achieve high performance and require no additional hardware [34]. Hence, major vendors, like Intel with SGX [22] and ARM with TrustZone [7], utilize this approach. However, sharing resources, such as caches or peripherals, also is the main disadvantage of such systems. SGX and TrustZone fail to provide strong isolation guarantees by being vulnerable to side-channel attacks, such as cache attacks [15, 27, 28, 36, 49, 65] or transient attacks [16, 20, 55]. Additionally, most TEEs, e.g., SGX, based on this design approach, do not protect peripherals by providing secure I/O.

**External Co-Processor based TEEs.** These TEEs utilize a dedicated, external core to securely execute trustlets. Examples of such secure elements are Google's Titan [32] and Samsung's eSE [45], which are mainly used in mobile platforms or servers. Since this technique separates the execution pipelines and the caches of REE and TEE, cache- and transient-based side-channel attacks are mitigated efficiently. Although these TEEs provide strong security guarantees, they also have major drawbacks. External secure elements typically establish a communication channel between REE and TEE using a slow communication interface, like SPI. This limits potential use cases of the TEE and the exposed interface also is vulnerable against physical probing and sniffing attacks [3]. Furthermore, dedicated secure co-processors require additional hardware and do not support shared peripherals.

**On-SoC Processor based TEEs.** This approach combines advantages from virtual processor and external co-processor based TEE designs. Here, a dedicated secure element is directly embedded into the main SoC infrastructure of the system. Although placing

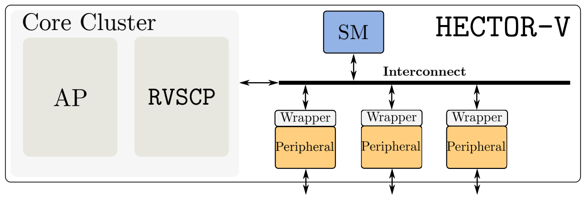

**Figure 1: Overall HECTOR-V design.**

such an element into the SoC increases the die size of the chip, this approach enables fast communication channels between REE and TEE and aggravates probing and sniffing attacks on this interface. Therefore, and due to the possibility to share peripherals on the SoC, this approach increases the flexibility and possible use cases of the design. Additionally, by placing two separate cores into the system, cache- and transient-based side-channel attacks can be mitigated. This approach is taken by major vendors, like Apple with their Secure Enclave Processor (SEP) [29] and Microsoft with their recently announced Pluton processor [61].

## 3 THREAT MODEL

Our threat model considers the common attack scenarios on TEEs defined by Cerdeira et al. [17]. We are considering an attacker directly exploiting architectural weaknesses of the TEE and the TEE kernel. Here an adversary uses a combination of bugs in the kernel, flaws in the hardware protection mechanism, and missing state-of-the-art defense strategies, such as ASLR or guard pages, to compromise the system. Furthermore, we expect bugs in trustlets, which can be exploited by an attacker over the communication interface between REE and TEE. The security of applications in REE or TEE also can be threatened by using cache-based or transient-based side-channel attacks. Additionally, we are considering a malicious trustlets explicitly trying to attack the SoC. We are extending the threat model of common TEEs also to cover attacks on peripherals, *i.e.*, illegitimate accesses to secure storage elements, sensitive peripherals such as a fingerprint reader or SPI [1], or protected memory regions. In summary, we expect a powerful attacker having full control over user applications, the operating system, trustlets, or even the hypervisor executed on the application processor.

## 4 DESIGN

This section presents our secure TEE design approach consisting of the HECTOR-V architecture and the secure processor RVSCP. We first introduce HECTOR-V consisting of several architectural features, such as trusted I/O paths, a security monitor, and a secure TEE integration. Then, we introduce RVSCP, a concrete proposal for a secure processor used as a TEE and demonstrate, how RVSCP and HECTOR-V combined, form a secure TEE system. In Section 5, we then demonstrate a prototype of HECTOR-V and RVSCP integrated into our RISC-V base platform, the lowRISC chip.

### 4.1 HECTOR-V Design

The HECTOR-V design proposes architectural features for creating secure TEEs. As seen in Figure 1, the TEE in HECTOR-V is realized by mounting a secure processor directly into the main SoC. This approach is similar to ARM's big.LITTLE technology [6], where independent cores are embedded into the chip. However, instead of using the additional cores for power efficiency, HECTOR-V uses the dedicated processor for security purposes. Although placing a secure processor directly into the SoC increases the area overhead, we argue that this approach is feasible and yields several advantages. First, we contend that offering strong isolation between secure and non-secure domain only is possible with a dedicated secure processor for the TEE. TEE architectures introduced before, such as Arm TrustZone and Intel SGX, are vulnerable to side-channel and microarchitectural attacks [19, 36]. As these attacks are the result of the growing complexity of modern processors driven by the demand for high-speed systems [34], fixing these security issues in the main processor is challenging [42]. However, the design choice of using a dedicated secure processor for the TEE completely separates the instruction pipelines and caches of the cores, resulting in an independent execution flow of REE and TEE applications. Second, services deployed on TEE processors, such as applications handling banking information or processing passwords, are typically rather small. This constrained processing requirement allow us to deploy a tiny secure processor. Compared to a state-of-art multicore processor, the area overhead introduced by this tiny processor is negligible. While integrating a dedicated secure processor into the SoC infrastructure is straightforward, a secure and viable TEE system requires to establish communication channels between REE and TEE, as well as for peripherals enabling user I/O. In the remaining part of this section, we introduce our trusted I/O path design utilizing a hardware-based security monitor and elaborate how these concepts allow REE and TEE to share devices on the SoC and realize features, such as secure storage elements.

*4.1.1 Trusted I/O Paths.* The trusted I/O path mechanism is the central element of the HECTOR-V architecture. This concept allows HECTOR-V to securely share devices on the SoC, establish secure communication channels between external users and the TEE or REE, and implement concepts, like secure storage elements. HECTOR-V establishes trusted I/O paths by assigning an unique identifier to each party of the system, *i.e.*, the processor cores. When accessing a device, like the SD-card or the block RAM (BRAM), the peripheral uses an internal mechanism to verify the identifier. This strategy allows the architecture to enforce access permissions, such as the fingerprint reader only can be accessed by the TEE.

**Identifier.** In HECTOR-V, an identifier is used to distinguish between legitimate and illegitimate accesses to a peripheral. While this concept is similar to ARM TrustZone [7], TZ only uses a 1-bit identifier to distinguish between accesses from the secure or non-secure world. The identifier used by our system consists of a core ID, a process ID, and a peripheral ID. While the core ID permanently and unchangeable is assigned to each processing core, e.g., an ID for the application processor and one ID for the secure processor, at design time, the process and peripheral ID can be assigned by each entity itself.

**Interconnect.** To efficiently transport the ID from the participants to the peripherals, we integrate the ID directly into the system interconnect. The AMBA AXI4 [5] protocol, which is used as interconnect by our lowRISC base platform and many other SoC designs, allows to embed up to 1024-bits wide user-defined signals into the protocol. By embedding the ID into the user-defined signals, which are not used by most AXI devices, the ID is transported without any protocol overhead. Additionally, this approach assures that the identifier is sent along with the address and data on each AXI request. Since the core ID is hardcoded directly into the the interconnect interface of the participants, an attacker cannot change this ID from software.

**Peripherals.** HECTOR-V creates a trusted channel between an entity, e.g., the application or the secure processor, and a peripheral by enforcing the identifier-based access check directly in hardware at the peripheral driver. The peripherals, e.g., the hardware implementation of the SD-card controller or the BRAM, use the ID to filter illegitimate accesses. This scheme, which requires the peripherals to be identifier-aware, is implemented by introducing a lightweight wrapper module for each peripheral instance. In our architecture, the peripheral wrapper module consists of two communication channels: the data channel and the configuration channel. The data channel interface of the peripheral wrapper is directly connected to the AXI4 communication fabric, allowing other parties, e.g., the secure processor, to access the peripheral. By using the configuration channel, the configuration party can set or unset the identifier in the ID field. In HECTOR-V, the data and configuration channels are physically separated by introducing a dedicated, lightweight AXI4-lite [5] communication fabric.

When the identifier of the AXI4 communication request transported over the communication fabric matches the identifier stored in the peripheral wrapper ID field, the access is permitted and the request is directly forwarded to the actual peripheral driver. On an ID mismatch, the firewall mechanism blocks the request and returns an exception over the AXI4 channel, which can be processed by the issuer. If the process or peripheral identifier is set to zero, the peripheral wrapper ignores these ID fields.

The peripheral wrapper resides in two states: claimed or unclaimed. While in the unclaimed state all data channel requests are blocked, in the claimed state only the requests with the matching identifier are permitted. Furthermore, we differentiate between configurable and non-configurable peripheral wrappers. To realize functionalities, like secure storage elements, which must not be accessible by any party except one, non-configurable peripheral wrappers store a hardcoded identifier, consisting of the core and process ID, in the identifier field.

*4.1.2 Security Monitor.* The architectural features of identifier, identifier transportation, and peripheral firewalls are the foundation for establishing trusted I/O paths on the SoC. To enforce security policies utilizing the trusted paths, we embed a tiny security monitor into the SoC infrastructure. This hardware-based security monitor (SM) module acts as trusted computing base (TCB) and is responsible for managing access permissions to peripherals. Internally, the SM consists of a table tracking these access permissions. For each peripheral included in the system, the security monitor maintains a table entry tracking the state (claimed or unclaimed) and a list of identifiers allowed to claim the device.

**Security Monitor Owner.** The HECTOR-V architecture introduces the security monitor owner privilege, which is assigned to a certain party at design time. This party solely is responsible for configuring the TCB, *i.e.*, the security monitor, over an interface. More concretely, only the SM owner is allowed to define, which peripheral can be claimed by which participant. The security monitor stores the identifier of the current owner and accepts configuration commands only from this party.

**Peripheral Claiming and Releasing.** To access a certain peripheral, the requester first needs to claim the peripheral by sending a claim request to the security monitor. Then, the SM checks, if the peripheral currently is unclaimed and verifies that the ID of the issuer is in the list of identifiers permitted to access the peripheral. Finally, the security monitor sends the ID of the issuer to the peripheral wrapper over the AXI4-lite bus and this wrapper sets this identifier in its ID field. Now, the identifier transported in the user-defined signals of each AXI4 request issued by the requester matches the identifier in the ID field and the requester exclusively can access the peripheral. However, if the requester is not allowed to claim the peripheral, *i.e.*, the identifier of the requester is not in the list of privileged entities, the ID verification fails and the security monitor sends back an access denied message. If the requester ID is in the table entry of the peripheral, but the peripheral currently is claimed, the security monitor notifies the issuer. HECTOR-V is a cooperative scheme, meaning, the entity currently claiming a peripheral needs to release it after using it. To release a peripheral, the entity sends a release command to the SM. The security monitor processes this request by clearing the ID field of the peripheral wrapper.

**Peripheral Access Withdraw.** Claiming a peripheral and binding access exclusively to an entity is a powerful concept and establishes a trusted, secure channel between this entity and a peripheral, but it also can be abused. A participant, e.g., the application processor or the secure processor, in control of an attacker, could permanently occupy one or more peripherals resulting in a denial-of-service (DOS) attack. To mitigate such attacks and manage unresponsive participants, the security monitor has the capability to withdraw access from certain peripherals. A simple approach to implement this functionality would be to clear the ID field in the peripheral wrapper. However, this approach is dangerous because it could enable time-of-check to time-of-use (TOCTOU) attacks. For example, when withdrawing access to a peripheral currently processing a secret, the SM owner would be able to retrieve the result of the request. To securely withdraw access to certain peripherals, HECTOR-V introduces a withdrawing mechanism, which can be triggered by any entity in the system. While a withdraw request issued by the SM owner is always granted, a request from other parties first needs to be approved by the security monitor. When a withdraw request is retrieved by the security monitor, the SM simultaneously starts a timer with a predefined timeout and notifies the owner of the peripheral. The notification of the peripheral owner is realized by introducing a dedicated interrupt line and a interrupt service routine (ISR) provided by the processors for each peripheral. The ISR implements a cleanup function responsible for clearing secrets, stopping current transactions, and gracefully releasing the corresponding peripheral. After the timeout is reached in the security monitor and the peripheral is not released gracefully by the ISR, the SM forcefully releases the peripheral by clearing the ID field in the firewall.

**Security Monitor Ownership Transfer.** A significant advantage of HECTOR-V, compared to other TEE architectures, is the possibility to utilize the TEE infrastructure for various use cases.

While in TEE systems, like ARM TrustZone, one entity, e.g., the secure-world, is the exclusive owner of the highest privilege level, HECTOR-V introduces a dynamic ownership transfer mechanism. The security monitor privilege, which allows the SM owner to configure access privileges and release arbitrary peripherals, can be transferred to any other entity by the SM owner. To initiate a SM ownership transfer, the entity sends a request with the identifier of the new owner to the security monitor. The security monitor acknowledges this request by setting the received identifier into the SM owner ID field. To obtain a clear state, we recommend that the security monitor owner first releases all peripherals.

## 4.2 RVSCP Design

Most ARM TrustZone based systems either utilize a dedicated TEE OS to spawn multiple trustlets or exclusively reserve the whole secure domain for one trustlet providing services [44]. While running multiple trustlets is tempting, the concept of deploying a fairly complex OS also increases the TCB and the attack surface. However, as reserving the whole TEE environment for a single trustlet might be lavish, we propose a TEE design in between of these two approaches. In RVSCP, we reduce the TCB to a minimum by providing basic OS services, such as context switches and resource management, directly in hardware. Although these hardware services cannot provide the same functionality as a rich OS, we argue, that for most TEE use cases, this approach is sufficient. Typically, trustlets deployed in TEEs offer services with limited complexity, such as a key store, a credential manager, or a cryptographic library [32, 38]. Furthermore this approach also allows us to deeply integrate security features, such as a control-flow integrity (CFI) unit combined with the secure I/O concept of HECTOR-V, into the processor.

### *4.2.1 Control-Flow Integrity.*

To protect a program from attacks targeting to alter the control-flow, CFI schemes are commonly used [53, 57, 64]. These schemes mitigate attacks like ROP [50] or JOP [18] by ensuring that the control-flow of the program can not escape the control-flow graph (CFG) determined at compile time. Enforcing the integrity of the instruction stream can be realized in different degrees of fineness. While some schemes [2, 8, 21] preserve the correctness of the execution flow at basic block level, other techniques [24, 59, 60] maintain the integrity of the control-flow even at instruction granularity.

**Control-Flow Integrity Unit.** RVSCP utilizes the existing fine-grain Sponge-Based Control-Flow Protection (SCFP) [59] scheme to protect trustlets within the secure processor from attack attempts. The main idea of SCFP is to encrypt a program using a sponge-based authenticated encryption primitive during compile time and decrypt it instruction for instruction at runtime. This method allows SCFP to enforce the, at compile time extracted, CFG at runtime. Decryption of the individual instructions is realized using a dedicated decryption stage in the processor pipeline. To successfully decrypt an instruction, the pipeline stage needs to know the key and an internal state. The SCFP scheme accumulates information over all previously executed instructions in this state. If the integrity of the state is violated, the decryption fails and returns a faulty instruction, which can be detected with a certain probability by the CPU. An attacker modifying instructions, e.g., using fault injection, or inserting additional instructions, alters the state and can be detected by SCFP.

### *4.2.2 Hardware Scheduling.*

We extend the native SCFP implementation, which allows to execute a bare-metal program on a processor, to support multitasking. One approach to enable multitasking with CFI protected trustlets could be realized purely in software using an OS. However, since TEE operating systems, such as OP-TEE [26], do not provide state-of-the-art protection mechanisms, such as ASLR or guard pages, mounting an operating system would also increase the attack surface of the TEE. Therefore, similar to Antikernel [66], RVSCP offers hardware features to run multiple trustlets on the processor and reduce the software TCB to a minimum.

**Hardware Scheduling Unit.** RVSCP introduces a hardware entity providing minimal OS functionality for trustlets. This hardware unit is responsible for performing secure context switches between individual trustlets in hardware. The round-robin based scheduling mechanism internally maintains a list of trustlets and after a certain amount of cycles executed, a context switch is conducted. When performing a context switch, the hardware entity stops the current trustlet, stores the architectural state to a secure place, and loads the next architectural state of the next trustlet. Additionally, the hardware context switch mechanism also exchanges the decryption key used for SCFP. Using an individual key for each trustlet yields two major advantages. First, since the programs are encrypted with a different key, only the developer with the correct key can access the plain program, leading to an IP protection mechanism for trustlets. Second, using different keys for trustlets enables strong isolation between the applications. After the context switch, the execution is resumed and the next trustlet is executed. While this hardware scheduling unit allows the processor to basically consist of several virtual processor cores, only a fixed number of trustlets can run simultaneously on the physical core. However, since most TEEs are anyway limited in their processing power and only a well-chosen set of trustlets is usually deployed in TEEs by the vendor, an upper bound of processes is acceptable. Furthermore, we argue that for simple services typically used in mobile platforms, such as a process handling biometric data for unlocking the device or a process handling the secure key storage, no dedicated operating system is needed. Completely omitting the operating system and providing operating system services using a tiny hardware unit reduces the software TCB to a minimum and would even allow to formally verify the simple hardware unit.

### *4.2.3 Control-Flow Integrity with Secure I/O.*

The fine granular control-flow integrity unit embedded into the HECTOR-V architecture prevents an attacker from performing control-flow hijack attacks by limiting the control-flow of the program to only valid paths through its control-flow graph. However, while the CFI scheme protects the control-flow by detecting integrity violation of forward- and backward-edges, and thus prevents attacks such as ROP or JOP, data-oriented attacks can not be detected by this countermeasure. In such attacks, an adversary modifies control- or non-control related data to break the security of the system. By manipulating control-related data, such as the condition value in an `if` statement, the attacker indirectly can influence the control-flow of the program. Furthermore, an attacker could leak sensitive data, such as passwords or keys, by manipulating addresses in the system. For example, instead of returning the ciphertext over an API to the user, an attacker could modify the address from pointing to the location of the ciphertext in memory to the encryption key stored in a secure storage element instead by exploiting a buffer overflow. To lower the impact of such attacks, RVSCP binds access to certain assets to a certain CFI state. More concretely, only when the CFI protected program reaches a predefined CFI state, the program is permitted to access the distinct peripheral. In RVSCP, this strategy is realized by tunneling each interaction request with a peripheral through a dedicated function with a certain state. RVSCP automatically sets the peripheral identifier of the processor to the current state. Only if this state matches the ID stored in the peripheral wrapper, access to the device is granted. If an attacker calls the peripheral access function outside the valid control-flow graph, the CFI mechanism detects this violation. Additionally, if the adversary crafts an address accessing the asset, the state used as an ID does not match the identifier of the peripheral and access to the device is restricted.

## 5 IMPLEMENTATION

In this section, we provide a prototype implementation of our HECTOR-V architecture consisting of two cores with two different ISAs. We first introduce the RISC-V lowRISC chip, which we use as our base platform. Then, we extend this platform with the HECTOR-V features. Finally, we present our RVSCP design and demonstrate how this processor is embedded into the SoC infrastructure.

### 5.1 Base Platform

The prototype implementation is based on the open-source lowRISC [37] project. Internally, the lowRISC chip consists of the 64-bit RISC-V Rocket chip [9, 58] using the RV64GC ISA. The SoC, capable of running Linux, provides an external off-chip DDR3 memory and an on-chip BRAM.

### 5.2 HECTOR-V

Figure 2 depicts the prototype implementation of HECTOR-V. The extended lowRISC base platform consists of the Rocket Core application processor 🚀 and the secure processor RVSCP integrated into the TEE ⚓. By providing an AXI4 master interface ① ②, REE and TEE gain access to various peripherals over the shared communication fabric ③. To differentiate between REE and TEE, the immutable core identifier is directly embedded into this AXI4 master interface. We further introduce a security monitor 🛡, a memory protection unit (MPU) 🗄, a reset unit ⏻, and a secure storage element 🔑 as part of HECTOR-V.

**Interconnect.** The SoC communication infrastructure consists of a shared AXI4 ③ and AXI4-lite Ⓒ interconnect. In our architecture, the AXI4 bus is used to enable interaction between the processing cores and the peripherals. We extended the AXI4 bus protocol and the crossbar to transport the 16-bit user signal along with each bus request. This user signal carries the 1-bit core ID, the 4-bit process ID, and the 10-bit peripheral ID. While communication between peripherals and cores requires a high-speed link, the configuration of the security monitor and the peripheral wrappers only consists of small configuration commands. Hence, we use a lightweight AXI4-lite bus as a configuration channel. The configuration of the security monitor is realized by introducing a point-to-point communication channel between REE and SM Ⓐ and TEE and SM Ⓑ. By using a separate AXI4-lite interconnect Ⓒ, the configuration of the peripheral wrappers can only be initiated by the security monitor. This strategy ensures that neither the TEE nor the REE directly can manipulate the peripheral firewalls; configuration only can be requested over the security monitor module.

**Security Monitor.** To receive commands from REE and TEE, the security monitor implements two AXI4-lite slave interfaces. The protocol used to interact with the security monitor consists of two privileged and four unprivileged commands. With the privileged configuration command, the SM owner configures the table entries of the hardware module. The table includes all peripherals known to the SM, the current claiming status, and a list of identifiers allowed to request access to the device. By using the privileged ownership transfer configuration command, the SM owner can define a new designated SM owner. The permission to issue a privileged configuration command is checked by the security monitor using the SM owner ID field stored in the SM module. The unprivileged commands consist of a claiming and release request command allowing the issuer to gain access to a peripheral. A status command can be used to determine the permission level of the peripheral and if it is currently claimed. To gain access to an already claimed peripheral, the unprivileged withdraw request command can be used. While a withdraw request issued by the SM owner is always granted, the request of the unprivileged participant first needs to be approved. When the withdraw request is granted, the SM uses dedicated interrupt lines 💣 to notify the owner of the peripheral about the withdraw request. For each peripheral, a dedicated interrupt line between SM and TEE or REE is introduced. When the request is approved by the SM, the configuration command is forwarded to the peripheral wrapper over the AXI4-lite crossbar Ⓒ.

**Peripheral Wrapper.** An AXI4 read or write request issued by the TEE or REE and transported over the interconnect ③ is not directly sent to the peripheral driver. First, a simple logic deployed in the peripheral wrapper checks if the ID stored in the user signal of the AXI4 request matches the identifier stored in the ID field of the wrapper module. When the process ID or the peripheral ID is set to zero, the access control is only conducted with the core ID, allowing all entities on the TEE or REE to access the peripheral. Then, if the ID transported in the request matches the ID stored in the module, the request is forwarded to the actual peripheral. However, on an ID mismatch, the peripheral wrapper transports the error code SLVERR to the issuer using the RRESP or BRESP AXI4 signal. In addition to the AXI4 slave interface, the peripheral wrappers also implement an AXI4-lite slave interface. This interface is used by the SM to set or unset the I in the ID field of the wrapper.

**Interrupt.** The peripheral wrapper also is responsible for routing the interrupt line of the peripheral to the current peripheral owner. To realize correct interrupt handling, the peripheral wrapper modules consists of one dedicated interrupt line for each participant (REE and TEE). Based on the ID of the current peripheral owner, the interrupt either is routed to the TEE or the REE.

**Software Support.** To interact with the security monitor, we provide a Linux kernel module for the application processor. This kernel module allows the user processes to claim, release, withdraw,

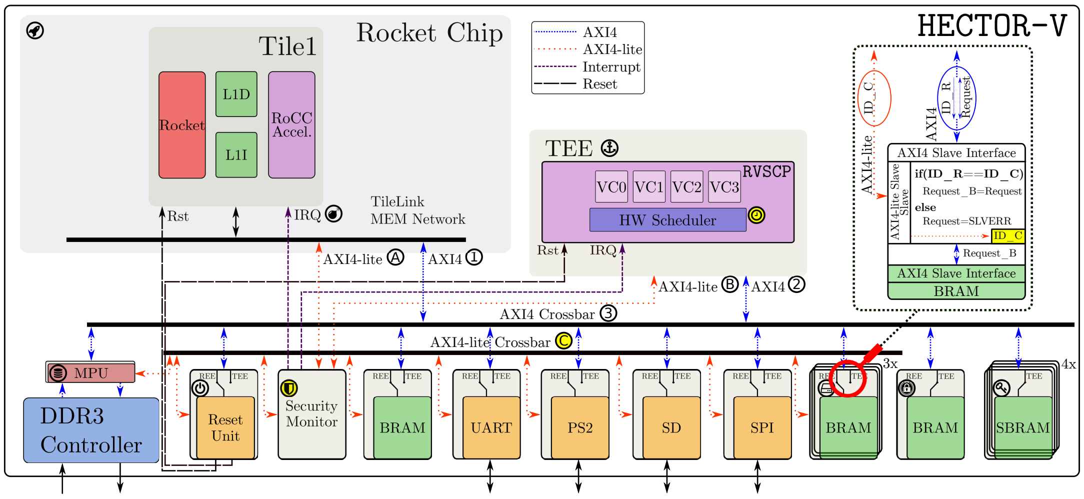

**Figure 2: The HECTOR-V architecture.**

or query the status of a peripheral. Furthermore, the kernel module also provides functionalities to configure the security monitor. To handle interrupts from the security monitor withdraw mechanism, each peripheral driver is extended with a dedicated cleanup interrupt service routine. This ISR is responsible for clearing any secrets, aborting communication channels with other parties, and releasing the peripheral gracefully using the release mechanism. For the TEE, we provide a small library. Similar to the kernel module, this library provides basic functionalities to interact with the security monitor

**Physical Memory Protection.** Isolating a peripheral by binding it exclusively to one entity does not work for the shared, external DDR3 memory. Therefore, the HECTOR-V architecture introduces a memory protection unit (MPU), which is placed directly between the memory controller and the AXI4 bus interface. The MPU can be claimed like any other peripheral by the TEE or REE using the security monitor and the dedicated AXI4-lite slave interface. The party currently claiming the MPU is now able to divide the physical memory into up to 16 regions. These regions can then be either exclusively accessed by one entity or are shared among multiple entities. Each incoming AXI4 read or write access is checked by the memory protection unit using the identifier transported with the request. With this mechanism, a secure storage place for the REE and each virtual core of the RVSCP is enabled and shared communication channels between REE and TEE can be established.

**Reset Unit.** The reset unit controls the reset lines of the application and the secure processor. Similar to other peripherals, this entity can be claimed by each participant in the system. The owner of the device can release the reset lines and turn on or off the other entity in the system.

**Secure Storage.** To securely store sensitive data, such as cryptographic keys, biometric data, or user passwords, HECTOR-V introduces a secure storage element. In contrast to other peripherals, a predefined, immutable identifier consisting of the core ID and the process ID is programmed into the ID field of the wrapper module at design time. Therefore, only the entity with the corresponding ID can access the storage element. In the prototype, each of the four virtual cores of RVSCP posses an own secure storage.

## 5.3 RVSCP

The RVSCP prototype implementation is based on the 32-bit RISC-V REMUS core [46, 59] with the RV32IMXIE ISA already offering the sponge-based CFI unit. The REMUS core originally is based on the Ri5CY core [54], which achieves similar performance than a ARM Cortex-M4 core. We further extend the core with the RVSCP features and embed the processor into the HECTOR-V architecture.

**TEE Infrastructure.** By using the AXI4 master interface ② connected to the AXI4 crossbar ③, the RVSCP is able to interact with the peripherals, such as the UART or PS2 controller. Similar to the main application processor, the secure processor implements an AXI4-lite master interface Ⓑ to configure and transmit peripheral access requests to the security monitor.

**Context Switch.** The hardware scheduler unit is responsible for performing secure context switches. For the RVSCP prototype, the hardware scheduler maintains a list of four slots enabling four virtual cores VC0 . . . VC3 on the RVSCP core. On a context switch, the hardware scheduler saves the current SCFP state and the current register file and loads the state, the decryption key, and the register file for the next trustlet. To implement the replacement of the register file, we added four additional register sets to our processor. Note that the register file replacement could also be implemented by storing and loading the content of the registers to memory, e.g., to a secure storage element, to keep the area overhead of the processor low. To differentiate between the four trustlets executed on the RVSCP, each of the four slots gets assigned an individual process ID. While the core ID is identical for all slots, the hardware scheduler replaces the process ID directly in the AXI4 and AXI4-lite master interface individually for each slot. By using the same core ID for all four threads, a peripheral could be configured to be accessible by all four trustlets.

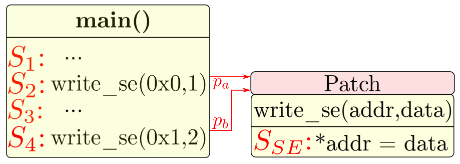

**Figure 3: Access function for secure peripheral accesses.**

**Decryption Keys.** To decrypt the encrypted instruction stream, the SCFP unit needs to know the decryption key for the corresponding trustlet. In the prototype implementation, the key is stored in a dedicated control and status register of each virtual core, which is only accessible from the respective core. To initially load the key into the secure storage, our prototype trustlet consists of a small, unencrypted boot code and the actual, encrypted code. The unencrypted boot code can either generate the key, load the key from the network, or directly from the binary. After storing the key into the key register, the actual encrypted trustlet starts to execute.

**Code Storage.** Each of the virtual processor cores `VC0` ... `VC3` running on the `RVSCP` processor executes code from an on-chip BRAM. While `VC1` ... `VC3` fetch code from a claimable BRAM 🖴, the first virtual processor core fetches its code from a secure code storage element 🔒 exclusively and permanently owned by `VC0`. To utilize one of the virtual cores at `RVSCP` as an enclave, the issuer needs to store the trustlet code on the BRAM of this core. Since the code of the first virtual core is stored in a secure storage element 🔒, this code cannot be changed by the REE or by the other virtual cores of `RVSCP` at any point in time.

**Control-Flow Integrity with Secure I/O.** To implement the concept of binding access to peripherals to a certain CFI state, each AXI4 request leaving the `RVSCP` is annotated with the current CFI state. The processor directly places a compressed form of the current state into the 10-bit peripheral ID field of the AXI4 user signal. To only allow, e.g., write access to the secure element when reaching a predefined state, the trustlet tunnels all write accesses to this peripheral through the function `write_se`. Since decrypting this function only works, when a certain state $S_{SE}$ is reached, the SCFP scheme automatically patches the state of the callee with the patch value $p_a$ or $p_b$. As seen in Figure 3, the patching mechanism of SCFP ensures that, on each valid call of the access function, state $S_{SE}$ is reached. By using this state as the peripheral identifier, access to a specific peripheral only is granted when reaching state $S_{SE}$ through the access function. Similar to the setup procedure of the decryption keys, the boot code of the trustlet claims the peripheral used in the code by sending the identifier consisting of the core identifier (ID of `RVSCP`), the process identifier (ID of the virtual processor), and the peripheral identifier (compressed CFI state $S_{SE}$) to the security monitor. Now, access to secrets stored in the secure storage element or other sensitive peripherals, like a fingerprint reader, only is permitted when reaching a predefined CFI state. If a trustlet does not need explicit protection of certain peripherals, the peripheral ID in the peripheral claiming request is set to zero.

## 5.4 Performance Evaluation

In this section, we analyze the latency for peripheral and TEE communication, as well as the runtime overhead of the TEE.

**Peripheral Access Latency.** The secure I/O design of `HECTOR-V` requires the entities to claim resources, e.g., the BRAM, before accessing them. However, this claiming process, which is initialized in software, only needs to be conducted once for each entity and resource. To determine the performance of this mechanism, we configured a trustlet on the `RVSCP` to be the designated security monitor (SM) owner. We measured an execution time of 188 cycles on average for sending a claim request from the REE to the SM and reading back this status. Furthermore, the secure I/O concept of `HECTOR-V` introduces the peripheral wrappers, which compare the identifier of each incoming AXI4 request to the identifier stored in the register of this wrapper. To measure the performance impact of this filtering mechanism, we copied data from the REE to a protected BRAM. However, since the ID comparison only requires a simple hardware logic, we could not measure any performance impact when comparing to a platform without the peripheral wrappers.

**Communication Latency between REE and TEE.** The current implementation of `HECTOR-V` utilizes a memory region shared between REE and the trustlet for communication. Hence, the communication latency between these two security domains is determined by the performance of the external DDR3 memory and the communication API arranged by the trustlet and the REE application. To characterize the general performance of the underlying memory subsystem, we benchmarked the external memory using LMBench [41]. We measured an average memory access latency of 876 ns for the `lat_mem_rd 64M 512` testcase. To measure the communication latency, *i.e.*, the timeframe of sending a message from REE to TEE and receiving the acknowledgment on the REE, we installed a benchmarking trustlet on `RVSCP`. We configured this trustled to establish a shared memory region on the external memory using the secure I/O design of `HECTOR-V`. This trustlet acknowledges the received 1 MB message by writing a status value in the shared memory. On the REE, we implemented a blocking function allowing to send data to the shared memory region and waiting for the acknowledgment from the trustlet. Note that the usage of blocking functions for communication also is recommended by the popular GlobalPlatform TEE API specification [52] used by many TEE operating systems, such as OP-TEE [26]. We measured an execution time of 5498 cycles on average for transmitting a 1 MB memory object to the TEE and receiving the response from the REE. For comparison, simply writing a 1 MB to the external memory already takes 5261 cycles on average. This timing difference of 4.5 % on average is negligible, as the memory performance is the major performance indicator for cross-domain communication. Although the usage of blocking functions is common for TEEs, `HECTOR-V` could further be extended to implement an interrupt-based mailbox system similar to Apple's SEP [38].

**`RVSCP` Runtime Overhead.** For the secure `RVSCP` processor, we extended the existing core implementing SCFP [59] with the hardware scheduler responsible for scheduling the trustlets and managing the identifiers. As these changes do not alter the single thread performance of the core, we observed same performance numbers for `RVSCP`. Werner et al. measured a performance overhead of 9.1 %

on average for protecting a broad range of macrobenchmarks with CFI. The baseline for these measurements is the Ri5CY [54] core, which achieves similar performance than a state-of-the-art ARM Cortex-M4 [47]. Hence, RVSCP is suitable for executing a vast variety of trustlets.

## 5.5 Area Overhead

To quantify the area overhead introduced by the additional RISC-V core and the HECTOR-V features, we synthesized the lowRISC base platform for a Xilinx Kintex-7 series FPGA. More concretely we synthesized the extended lowRISC platform with the additional HECTOR-V features and a selection of different RISC-V cores used as TEE processor. Table 1 shows the total hardware overhead of the HECTOR-V architecture consisting of either the Ri5CY [54], the REMUS [46, 59], or the FRANKENSTEIN [48] core used as secure TEE processor. The REMUS core, which is based on the Ri5CY, implements a decryption unit to offer the Sponge-Based Control-Flow Protection (SCFP) mechanism. FRANKENSTEIN, which is an extended version of the REMUS core, is a 64-bit RISC-V processor including the SCFP scheme and a pointer protection scheme for secure memory accesses. While the Rocket Core used in the lowRISC platform is capable of running Linux, the processor core still is rather small compared to application processors from vendors like Intel, ARM, and AMD. Therefore, when using a larger processor in the HECTOR-V architecture, the relative overhead introduced by the additional secure TEE processor is negligible. In Table 2, we list area number for different components in the HECTOR-V architecture using the RI5CY core as secure TEE processor.

## 6 USE CASES

This section proposes use cases for the secure TEE design consisting of the HECTOR-V architecture and the RVSCP. More concretely, we demonstrate how the TEE can be used to boot Linux on the main application processor securely. After the secure boot use case, we show how the RVSCP can be utilized to securely execute trustlets. Note that these two scenarios are only a selection of many other use cases that can be realized with HECTOR-V.

**Table 1: Lookup table (LUT) overhead of the overall architecture consisting of the HECTOR-V features and an additional secure processor compared to the lowRISC base project.**

| Configuration | Area [LUTs] | Area Overhead [%] |
|---|---|---|
| lowRISC base platform | 55,443 | - |
| HECTOR-V with RI5CY | 63,648 | 14.8 |
| HECTOR-V with REMUS | 67,024 | 20.89 |
| HECTOR-V with FRANKENSTEIN | 68,746 | 23.99 |

**Table 2: Hardware overhead of different features.**

| Component | Area [LUTs] | Area [%] |
|---|---|---|
| Rocket Chip | 33,341 | 52.38 |
| RI5CY | 5,780 | 9.08 |
| Security Monitor | 446 | 0.7 |
| Peripheral Wrapper | 43 | 0.07 |
| AXI4 Crossbar | 3,052 | 4.79 |
| AXI4-lite Crossbar | 93 | 0.14 |

## 6.1 Secure Boot

On the unmodified lowRISC platform, a zero stage bootloader (ZSBL) permanently stored in the on-chip BRAM first loads the Berkeley boot loader (BBL) from the external SD-card. Then, the ZSBL hands over control to the BBL first stage bootloader (FSBL). The BBL, which is linked against the Linux kernel, fetches the Linux image and sets up the environment by configuring the hardware threads (HARTS) and the memory. Finally, the Linux operating system starts. However, loading the FSBL and the Linux image directly from the SD-card to boot up the device is problematic in many ways. An attacker with physical access to the device can boot arbitrary code by simply modifying the unprotected boot files stored on the SD-card. This attack methodology also can be used in an online attack by overwriting the boot files. To only allow authenticated software to boot, vendors frequently embed a secure boot mechanism into their systems. Here, a chain-of-trust is generated by authenticating each boot file before executing it.

In our use case implementing secure boot using features of the HECTOR-V architecture, the first virtual core VC0 of RVSCP is the designated security monitor owner. At design time, the reset unit of the system is configured to release the reset line of RVSCP and keep the REE processor halted when applying power to the SoC. Additionally, the security monitor owned by VC0 is unconfigured, except for the reset unit which is claimed by the SM owner. When applying power to the device, VC0 starts to execute the ZSBL stored in the secure code storage 🔒. Since the secure code storage is permanently and exclusively owned by VC0, the ZSBL is isolated from the other parties and only VC0 can update this code using an update mechanism. Due to these strong protection mechanisms, the ZSBL stored in the secure code storage is the system root-of-trust. The ZSBL code first claims the SD-card controller and configures the memory protection unit of the DDR3 memory. Then, the ZSBL fetches the BBL from the SD card and stores it to the external memory. Before passing control to the BBL, the ZSBL determines the hash value of the loaded BBL image. When this hash value does not match the expected hash value stored in the secure storage element 🔑 of VC0, the boot process is aborted. Again, an update of the expected hash value only can be initiated by the first virtual processor core of RVSCP because the secure storage element exclusively is owned by this party and can not be claimed by any other party at any point in time. If the verification of the BBL was successful, the ZSBL releases the SD-card driver, gives the main application processor access to the DDR3 memory region by configuring the MPU, and triggers the reset unit to start the REE. Now, the Rocket Core starts to execute the modified BBL from the external memory. The modified BBL then requests access to the SD-card controller, loads the Linux image from the SD-card, and verifies the loaded image by comparing the computed hash value with the expected hash value. Finally, if the verification process succeeds, the Linux operating system starts and can claim the first peripherals by sending requests to the security monitor. Since each stage of the boot process now is cryptographically verified, the system is in a genuine state. Now, the VC0 passes the SM owner privilege to the main application processor (AP). This change is initiated by sending the identifier of the AP to the SM using the ownership transfer command. Although the REE is now in full control of the system, e.g., is able to switch of the secure processor using the reset unit, secrets stored in the secure storage 🔑 and the secure code storage 🔒 are still protected from AP accesses.

## 6.2 Trustlets

In this use case, we utilize the virtual processor VC1 of RVSCP to execute a trustlet. We compile the trustlet with a LLVM-based toolchain supporting the SCFP scheme and encrypt the program with the key $K$. To interact with the outside and to store secrets, the trustlet uses the UART controller and the secure storage element. The secure storage element is protected from malicious accesses by tunneling all requests through a dedicated access function and binding the access to it to the CFI state $S_{SE}$. We define access to the UART controller as uncritical. Therefore the interaction with this controller is unprotected. First, the application processor switches off the RVSCP by utilizing the reset unit. Then, the processor claims a claimable BRAM ⊜ and stores the code, consisting of an unencrypted small boot code and the encrypted trustlet, to this storage. After the code is saved, the AP sets VC1 as owner of the code storage in the security monitor. Furthermore, the MPU is configured to provide a memory region owned by the trustlet to use as RAM and a shared memory region to establish a communication channel between REE and the trustlet. Finally, the RVSCP is started by releasing the reset line of the secure processor and code gets executed. To initialize key $K$ and state $S_{SE}$, the boot code of the trustlet writes the decryption key into the key register of $VC1$ and claims the secure key storage element by setting the compressed state $S_{SE}$ into the peripheral ID field. Now, control is passed to the SCFP protected trustlet code and the decryption stage of the processor decrypts the code using the key. To access the UART controller, the trustlet needs to claim the controller by sending a claim request to the SM. The secure storage element is accessed by using the dedicated access function. By setting the compressed state directly into the peripheral ID field of each AXI4 request leaving the processor, access to the secure storage element only is permitted when reaching the predefined state $S_{SE}$. The communication channel established using the shared, external memory allows the REE and trustlet to exchange information. When the designated SM owner, the AP, withdraws access to the UART controller, the claimable BRAM, or the shared memory region, the trustlet is notified using an interrupt. This interrupt is handled by the ISR implemented on the trustlet. The ISR clears all secrets, aborts communication with the UART controller, gracefully releases the peripheral, and enters a predefined IDLE state.

## 7 SECURITY DISCUSSION

In this section, we analyze security guarantees of HECTOR-V and discuss security properties of the secure RVSCP processor.

### 7.1 HECTOR-V

The security guarantees of HECTOR-V are established with the introduced architectural features, *i.e.*, the security monitor, the peripheral wrappers, and the separation of the REE and the TEE in the SoC.

**Isolation between REE and TEE.** In each TEE design, the key challenge is to guarantee strong isolation between REE and TEE. Most TEE designs fail to provide this guarantee by either be vulnerable to side-channel and microarchitectural attacks or by insufficiently protecting TEE resources, such as the memory or peripherals [34]. HECTOR-V mitigates all cache- and microarchitectural-based side-channel attacks by separating REE and TEE using dedicated cores for each domain. This approach ensures that no sensitive components, such as caches, branch predictors, and the execution pipelines, are shared between REE and TEE. To protect resources, HECTOR-V provides architectural features to bind a peripheral to a entity using the SM and the peripheral wrappers. In HECTOR-V, for each entity, *i.e.*, the REE or the trustlets on the TEE, a dedicated memory region can be reserved. This memory region is isolated using hardware-enforced access checks in the MPU using the ID of the entity.

**Trusted Computing Base.** The TCB of HECTOR-V consists of several hardware and software features, which are marked in yellow in Figure 2. The central trust anchor is the security monitor ◐, which manages access permissions to peripherals and the memory. Here, the SM configures the peripheral wrappers over the exclusively owned AXI4-lite Ⓒ interconnect by setting the ID of the authorized entity into the ID register. In HECTOR-V, the SM only can be configured by one entity, the SM owner. This entity, the software TCB, either is a trustlet in the RVSCP or a kernel task in the REE and is responsible for configuring the security monitor. The SM differentiates between the SM owner and the other parties using the IDs. While the REE ID is permanently and immutable programmed into the AXI4 interface ① of this core, the identifiers of the trustlets in RVSCP are managed by the hardware scheduler ◎ TCB.

**Malicious Software TCB.** Although the SM owner is responsible for configuring the SM and is, therefore, part of the software TCB, HECTOR-V considers this entity as potentially malicious and provides architectural features to limit the impact of this threat. A malicious SM owner either could be the result of an attacker with kernel privileges on the REE currently being the SM owner or a malicious trustlet in the TEE with the SM ownership. Here, the goal of the attacker is to threaten the confidentiality, integrity, and availability of the system by misconfiguring the SM. HECTOR-V maintains the integrity and confidentiality of secrets stored in peripherals, e.g., the external memory or internal BRAMs, by exclusively binding these elements to one entity. The hardware-enforced ID access check prevents all other entities, even the SM owner, to access these elements. Furthermore, to protect critical secrets, such as keys, hash values, or the secure boot code, HECTOR-V offers the possibility to set an ID of an entity immutable into the ID field of the critical element, preventing the SM owner and other parties from accessing this element. Although the SM owner can initiate a peripheral release request, this hardware mechanism first notifies the current owner of the peripheral about the incoming withdraw procedure, allowing to clear sensitive data. However, a malicious

SM owner could influence the availability of the system by permanently withdrawing peripherals. To prevent the SM owner to switch off a different entity in the system and then access secrets stored in the peripherals, the reset unit could, similar to the withdrawing mechanism, notify the entity about the incoming reset.

### 7.2 RVSCP

The RVSCP processor, which is tightly integrated into the HECTOR-V architecture, consists of several hardware features, such as a CFI unit coupled with secure I/O and a hardware scheduler.

**Isolation between Trustlets.** In addition to the isolation guarantees between REE and TEE, the trustlets within the secure processor also need strong isolation between each other. The hardware scheduler, which acts as TCB for the RVSCP, is responsible of scheduling the trustlets. Here, on each context switch, this unit replaces the identifiers of each trustlets. As this swapping mechanism is realized in hardware, the trustlets cannot influence the ID in software. Isolating resources, such as the external memory, code storage, and peripherals, is realized by claiming separate resources for each trsutlet using the identifier-based secure I/O mechanism. While the heterogeneous architecture prevents an attacker from performing cache and transient-based attacks between REE and TEE, RVSCP needs additional countermeasures, such as flushing the microarchitectural state [63] on a context switch, to also protect against cache attacks. Due to the simplicity and openness of RVSCP, transient-based attacks cannot be performed by an adversary.

**Communication Interface.** To establish a communication between REE and TEE, a shared memory region for the trustlet and the REE application is created using the MPU and the ID of both parties. Recent attacks [11, 12] demonstrated that a single bug, e.g., a buffer overflow, in the software interface between REE and TEE could completely compromise the secure domain. To prevent the exploitation of such vulnerabilities, the trustlets are protected using hardware features of RVSCP. The CFI unit prevents an attacker from performing control-flow hijacking attacks, such as ROP or JOP. Furthermore, RVSCP limits the impact of data-oriented attacks, which cannot be detected by CFI units. Here, RVSCP protects trustlets and resources from these attacks by restricting access to peripherals only when reaching a predefined CFI state. Moreover, the attack surface of RVSCP is minimized by completely omitting a TEE OS and using a hardware TCB controlling the context switches and the resource management. In addition to logical attacks on the interface, the HECTOR-V approach also aggravates physical bus probing and sniffing attacks on the communication interface [3] by integrating the TEE into the SoC.

## 8 RELATED WORK

This section summarizes TEEs introduced by both, academia and industry, and compares them to HECTOR-V.

**Intel SGX.** In Intel Software Guard Extensions [22], an enclave is created purely in software and protected with hardware features, such as a memory encryption engine (MEE). Although this approach allows to flexibly spawn new enclaves in software, SGX suffers from several disadvantages. First, Intel explicitly excludes side-channel and fault attacks in their threat model [30] and this weakness already is actively exploited [15, 16, 20, 27, 43, 49, 55]. Second, SGX does not natively protect user I/O and is therefore vulnerable to attacks on peripherals.

**ARM TrustZone.** In this TEE design, ARM enforces a secure and non-secure domain within the processor using a virtual processor approach based on an identifier, the non-secure (NS) bit. The enforcement of the security policy is directly implemented in the AXI interface of the SoC components. This security policy can be static for some peripherals, like the fingerprint reader, or can be dynamically configured by the secure world using several hardware blocks [44]. The TrustZone Protection Controller (TZPC), which solely can be configured by the secure world, acts as a root-of-trust and manages access permissions for single peripherals [51]. To partition memory into secure and non-secure areas, the secure world can configure the security policies using the TZ Address Space Controller (TZASC) and the TZ Memory Adapter (TZMA). Since operating systems deployed in TrustZone typically do not provide state-of-the-art defense strategies [17], a malicious trustlet could threaten the security of the secure world. Therefore, vendors integrating TZ into their products limit their systems to only a pre-defined set of trustlets. SANCTUARY [14] tackles this problem by extending TrustZone to support user-space applications. Here, SANCTUARY utilizes TZ to exclusively reserve a core and memory to spawn individual Sanctuary instances. These instances, which allow users to deploy own trustlets, mitigate cache-based side-channel attacks by using the L1 cache of the reserved core and excluding to use the L2 cache. Although SANCTUARY enhances TrustZone to support user-defined trustlets, it inherits limitations of the underlying architecture. In contrast to HECTOR-V, TrustZone only uses a 1-bit ID to differentiate between secure and non-secure world. Hence, TrustZone only can bind peripherals to a core or to the secure domain and fails to offer fine-granular protection of peripherals for multiple trustlets [10]. Furthermore, TrustZone is an inflexible approach with the secure domain as the static trusted entity. HECTOR-V allows to dynamically switch the trusted domains between the entities, which enhances potential use cases for the TEE design. Additionally, HECTOR-V offers hardware features, such as a fine-granular CFI unit combined with secure I/O, to isolate trustlets within the secure domain. Although SANCTUARY mitigates cache-based side-channel attacks by reserving a core and its L1 cache for a trustlet, other ARM TrustZone based security architecture are still vulnerable against such attacks [36].

**RISC-V based TEEs.** The introduction of several open-source RISC-V cores mobilizes research on open TEE solutions. SANCTUM [23] aims to offer a similar programming model to SGX on RISC-V. This design uses a software-based security monitor and requires minimal hardware changes to dynamically spawn new enclaves. In addition to SGX, SANCTUM mitigates cache-based side-channel attacks by using cache partitioning for each enclave. Similarly, Keystone [35] utilizes a security monitor to enforce TEE guarantees and uses the physical memory protection (PMP) feature of RISC-V to isolate individual enclaves. To mitigate software side-channel attacks, Keystone flushes enclave states on a context switch. Although these designs address the side-channel problematic of TrustZone and SGX, SANCTUM and Keystone do not provide architectural features for secure I/O, leaving communication with peripherals unprotected.

**SiFive WorldGuard.** Concurrent developed with the HECTOR-V architecture, SiFive recently introduced WorldGuard [62]. Here, each core gets assigned a world ID and each process on the core can be annotated with a process ID. This ID then is transported using the interconnect and requests from participants are filtered by peripherals, the memory, and the caches. Similar to HECTOR-V, this approach allows WorldGuard to reserve one core exclusively for the secure domain to provide strong isolation guarantees. However, both architectures differ in various design choices. First, HECTOR-V uses a hardware-based security monitor which only can be configured by one party. In contrast to WorldGuard, the security monitor ownership can be dynamically transferred to other parties allowing flexible use cases. Additionally, the security monitor allows each participant in HECTOR-V to request access to certain peripherals and request access to already claimed peripherals using a withdraw request. We further propose a concrete secure processor design utilizing features of the heterogeneous architecture to create a trusted execution environment. Moreover, we comprehensively describe the hardware-software interaction and demonstrate features of HECTOR-V by introducing several use case scenarios.

**GPU based TEEs.** In addition to the proprietary solutions from Apple, Microsoft, and SiFive, several academic TEEs based on GPUs have been introduced recently. HIX [31] and Graviton [56] utilize the GPU to establish a trusted execution environment for trustlets. Although these designs share the idea of HECTOR-V to use a heterogeneous system for the TEE, these GPU-based architectures do not support secure I/O for peripherals.

## 9 CONCLUSION

In this paper, we proposed HECTOR-V, a secure TEE design strategy using a heterogeneous CPU architecture. Our design establishes secure paths between peripherals and the cores by tagging each party with an identifier. The peripherals enforce access permissions by checking the ID, which is transported along with each bus request. To configure these access permissions, we integrate a hardware-based security monitor into the architecture. The security monitor, which exclusively can set permissions, is owned by a configuration party. By allowing to transfer this ownership to other parties, HECTOR-V allows flexible permission management. In contrast to similar design approaches, we provide a notifier-based mechanism to withdraw access to certain peripherals securely. We further introduce RVSCP, a security-hardened CPU design tailored for our architecture. RVSCP combines a fine-granular control-flow integrity scheme with the secure I/O concept of HECTOR-V to restrict access to assets. To complete our TEE design, we introduce secure data and code storage elements, a reset unit, and a memory protection unit. We examine the features of our architecture in a secure boot and enclave scenario.

## ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402) and by the Austrian Research Promotion Agency (FFG) via the competence center Know-Center (grant number 844595), which is funded in the context of COMET - Competence Centers for Excellent Technologies by BMVIT, BMWFW, and Styria.

## REFERENCES

[1] 2018. CVE-2016-10423.

[2] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2009. Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.* 13 (2009).

[3] Denis Andzakovic. 2019. *Extracting Bitlocker Keys from a TPM.*

[4] Apple. 2020. *About the Apple T2 Security Chip.*

[5] ARM. 2019. AMBA AXI and ACE Protocol Specification. *arm.com* (2019).

[6] ARM. 2020. *Processing Architecture for Power Efficiency and Performance.*

[7] Architecure ARM. 2009. Security technology building a secure system using trustzone technology (white paper). *ARM Limited* (2009).

[8] Divya Arora, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. 2006. Hardware-Assisted Run-Time Monitoring for Secure Program Execution on Embedded Processors. *IEEE Trans. Very Large Scale Integr. Syst.* 14 (2006).

[9] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. The Rocket Chip Generator. *EECS Department, University of California, Berkeley, Technical Report* (2016).

[10] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. *arXiv abs/2010.15866* (2020).

[11] Gal Beniamini. 2016. *QSEE privilege escalation vulnerability and exploit (CVE-2015-6639).*

[12] Gal Beniamini. 2016. *War of the Worlds - Hijacking the Linux Kernel from QSEE.*

[13] Swapnil Bhartiya. 2020. *Linux in 2020: 27.8 million lines of code in the kernel, 1.3 million in systemd.*

[14] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves. In *Network and Distributed System Security Symposium – NDSS.*

[15] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *Workshop on Offensive Technologies – WOOT.*

[16] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security Symposium.*

[17] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA.*

[18] Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, and Marcel Winandy. 2010. Return-oriented programming without returns. In *Conference on Computer and Communications Security – CCS.*

[19] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. *arXiv abs/1802.09085* (2018).

[20] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. 2020. SgxPectre: Stealing Intel Secrets From SGX Enclaves via Speculative Execution. *IEEE Secur. Priv.* 18 (2020).

[21] Nick Christoulakis, George Christou, Elias Athanasopoulos, and Sotiris Ioannidis. 2016. HCFI: Hardware-enforced Control-Flow Integrity. In *Conference on Data and Application Security and Privacy – CODASPY.*

[22] Intel Corporation. 2019. Intel 64 and IA-32 Architectures Software Developer's Manual. (2019).

[23] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium.*

[24] Ruan de Clercq, Johannes Götzfried, David Übler, Pieter Maene, and Ingrid Verbauwhede. 2017. SOFIA: Software and control flow integrity architecture. *Comput. Secur.* 68 (2017).

[25] Jan-Erik Ekberg, Kari Kostiainen, and N. Asokan. 2014. The Untapped Potential of Trusted Execution Environments on Mobile Devices. *IEEE Secur. Priv.* 12 (2014).

[26] Trusted Firmware. 2020. *OP-TEE.*

[27] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017, Belgrade, Serbia, April 23, 2017.*

[28] Roberto Guanciale, Hamed Nemati, Christoph Baumann, and Mads Dam. 2016. Cache Storage Channels: Alias-Driven Attacks and Verified Countermeasures. In *IEEE Symposium on Security and Privacy – S&P.*

[29] Apple Inc. 2020. *Security enclave processor for a system on a chip.* US8832465B2.

[30] Intel. 2017. *Intel SGX and Side-Channels.*

[31] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *Architectural Support for Programming Languages and Operating Systems – ASPLOS*.

[32] Scott Johnson, Dominic Rizzo, Parthasarathy Ranganathan, Jon McCune, and Richard Ho. 2018. Titan: enabling a transparent silicon root of trust for Cloud. In *Hot Chips: A Symposium on High Performance Chips*.

[33] Sanjeev Khushu and Wilfred Gomes. 2019. Lakefield: Hybrid cores in 3D Package. In *2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, August 18-20, 2019*.

[34] Kari Kostiainen, Aritra Dhar, and Srdjan Capkun. 2020. Dedicated Security Chips in the Age of Secure Enclaves. *IEEE Secur. Priv.* 18 (2020).

[35] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: an open framework for architecting trusted execution environments. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*.

[36] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security Symposium*.

[37] LowRISC. 2019. *lowRISC Chip*.

[38] Tarjei Mandt, Mathew Solnik, and David Wang. 2016. Demystifying the secure enclave processor. *Black Hat Las Vegas* (2016).

[39] Steve McConnell. 2004. *Code complete*. Pearson Education.

[40] Elliot H. MednickEdward McLellan. 2020. Instruction subset implementation for low power operation. *US10698472B2* (2020).

[41] Larry W. McVoy and Carl Staelin. 1996. lmbench: Portable Tools for Performance Analysis. In *USENIX Annual Technical Conference*.

[42] Erik Kraft Michael Schwarz. 2019. Are Microarchitectural Attacks still possible on Flawless Hardware? *RuhrSec* (2019).

[43] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*.

[44] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51 (2019).

[45] Samsung. 2020. *eSE Safeguard against digital attacks*.

[46] David Schaffenrath. 2016. Fault-Attack Secure Processor Design. *Graz University of Technology* (2016).

[47] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. 2017. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*.

[48] Robert Schilling, Mario Werner, Pascal Nasahl, and Stefan Mangard. 2018. Pointing in the Right Direction - Securing Memory Accesses in a Faulty World. In *Annual Computer Security Applications Conference – ACSAC*.

[49] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware & Vulnerability Assessment – DIMVA*.

[50] Hovav Shacham. 2007. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *Conference on Computer and Communications Security – CCS*.

[51] José Alberto Moreira Silva. 2019. Arm TrustZone: evaluating the diversity of the memory subsystem. (2019).

[52] GlobalPlatform Device Technology. 2020. *TEE Client API Specification*.

[53] Caroline Tice, Tom Roeder, Peter Collingbourne, Stephen Checkoway, Úlfar Erlingsson, Luis Lozano, and Geoff Pike. 2014. Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM. In *USENIX Security Symposium*.

[54] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K Gurkaynak, and Luca Benini. 2016. PULPino: A small single-core RISC-V SoC. In *3rd RISCV Workshop*.

[55] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking transient execution through microarchitectural load value injection. In *41th IEEE Symposium on Security and Privacy (S&P'20)*.

[56] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *USENIX Symposium on Operating Systems Design and Implementation – OSDI*.

[57] Zhi Wang and Xuxian Jiang. 2010. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. In *IEEE Symposium on Security and Privacy – S&P*.

[58] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. 2011. *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA*. Technical Report. EECS Department, University of California, Berkeley.

[59] Mario Werner, Thomas Unterluggauer, David Schaffenrath, and Stefan Mangard. 2018. Sponge-Based Control-Flow Protection for IoT Devices. In *European Symposium on Security and Privacy – EuroS&P*.

[60] Mario Werner, Erich Wenger, and Stefan Mangard. 2015. Protecting the Control Flow of Embedded Processors against Fault Attacks. In *Smart Card Research and Advanced Applications – CARDIS*.

[61] David Weston. 2020. *Meet the Microsoft Pluton processor – The security chip designed for the future of Windows PCs*.

[62] Bob Wheeler. 2019. SIFIVE SECURES RISC-V. *Microprocessor report* (2019).

[63] Nils Wistoff, Moritz Schneider, Frank K. Gürkaynak, Luca Benini, and Gernot Heiser. 2020. Prevention of Microarchitectural Covert Channels on an Open-Source 64-bit RISC-V Core. *arXiv abs/2005.02193* (2020).

[64] Mingwei Zhang and R. Sekar. 2013. Control Flow Integrity for COTS Binaries. In *USENIX Security Symposium*.

[65] Ning Zhang, He Sun, Kun Sun, Wenjing Lou, and Yiwei Thomas Hou. 2016. CacheKit: Evading Memory Introspection Using Cache Incoherence. In *European Symposium on Security and Privacy – EuroS&P*.

[66] Andrew D. Zonenberg and Bülent Yener. 2016. Antikernel: A Decentralized Secure Hardware-Software Operating System Architecture. In *Cryptographic Hardware and Embedded Systems – CHES*.