Lightweight Security Architecture Based on Embedded Virtualization and Trust Mechanisms for IoT Edge Devices

Ramão Tiago Tiburski, Carlos Roberto Moratelli, Sérgio F. Johann, Marcelo Veiga Neves, Everton de Matos, Leonardo Albernaz Amaral, and Fabiano Hessel

Abstract

Security issues represent the greatest obstacle to the growth of edge computing and the Internet of Things (IoT). In this paradigm, IoT applications are migrating to edge devices. As a result, potential security risks are arising, and unauthorized access to data from IoT edge devices is becoming a real concern. Thus, there is a need for a comprehensive, end-to-end security approach since the system's more vulnerable point determines its overall security level. An edge device security system has to be built with a root of trust (i.e., something that is unclonable) and a chain of trust. Additionally, a security layer is necessary to ensure that different IoT applications execute isolated from each other in the device. In this sense, this article defines a security architecture that integrates trust mechanisms with embedded virtualization, providing security from hardware to applications. Our experiments show that the proposed architecture can be implemented with a smaller overhead and memory footprint compared to other proposed approaches in the literature, which makes it highly suitable for resource-constrained edge devices.

Introduction

The evolution of the Internet of Things (IoT) and the significant amount of data that has been exchanged between devices and the cloud have pushed the horizon to the edge computing paradigm [1]. It extends the cloud-based infrastructure to the edge of the network by increasing data processing and decision making on IoT edge devices, allowing more efficient communication with intermediary nodes [2]. However, security is emerging as one of the most significant challenges for edge computing [3]. The decentralization of IoT applications to the edge has made the devices more visible to attacks [4]. While some existing solutions in the context of cloud computing may address many security issues at the edge of the network, edge computing is introducing new security concerns due to its distinct characteristics [2].

The existing security paradigm will no longer be adequate for addressing the new security challenges in emerging devices [1]. According to [3], edge devices are more than data sources in an edge architecture. They are expected to preprocess data (e.g., aggregation, filtering) and make decisions, which makes them the primary targets of attacks [4]. Also, edge devices are often deployed in resource-constrained environments without strict monitoring and protection, thereby facing all kinds of security threats [3, 4]. Once an attacker has control of the device, no higher security mechanism can identify this condition, and all system execution and state can be compromised. According to [1], to increase the trust in such devices is the primary challenge. Hence, some aspects must be considered for security improvements: hardware-to-software oriented security and simple architecture building blocks for sustaining the system elements of the IoT.

This article promotes the use of embedded virtualization as an important choice to mitigate security challenges imposed by the edge computing paradigm [1, 2]. We propose a security architecture that integrates trust mechanisms and a hypervisor specially designed for resource-constrained devices. The novel aspect of the architecture is to increase the security of such devices with a smaller overhead and memory footprint than provided by existing approaches. A secure boot is used to enforce a root of trust environment and to build a chain of trust. Afterward, a trustworthy virtualization layer is booted up and used to initialize secure virtual machines (VMs). In our approach, intrinsic virtualization characteristics ensure protection during boot and runtime states, starting with the hardware platform and ending at IoT applications running on the device.

In the next sections, we present the problem statement and a security overview. We define the security architecture and present its evaluation regarding footprint, performance, latency, and security. We conclude the article by comparing our approach with existing works.

The authors define a security architecture that integrates trust mechanisms with embedded virtualization, providing security from hardware to applications. Their experiments show that the proposed architecture can be implemented with a smaller overhead and memory footprint compared to other proposed approaches in the literature, which makes it highly suitable for resource-constrained edge devices.

Ramão Tiago Tiburski, Sérgio F. Johann, Marcelo Veiga Neves, and Fabiano Hessel are with the Pontifical Catholic University of Rio Grande do Sul; Carlos Roberto Moratelli is with Federal University of Santa Catarina; Everton de Matos is with Pontifical Catholic University of Rio Grande do Sul and Meridional Faculty; Leonardo Albernaz Amaral is with FTEC Faculdade de Tecnologia.

Digital Object Identifier: 10.1109/MCOM.2018.1701047

0163-6804/19/$25.00 © 2019 IEEE

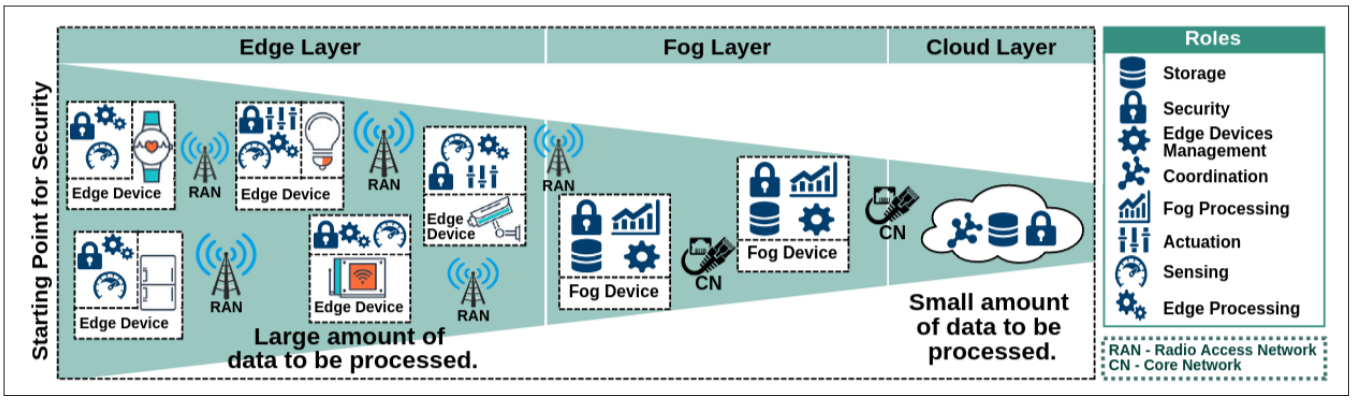

Figure 1. IoT architecture in three layers: edge, fog, and cloud.

Problem Statement and Security Overview

Edge computing takes place on IoT end devices, which means that less data is sent to the cloud [3]. Hence, IoT edge devices are becoming the most critical part of the architecture and the starting point for security [4]. Figure 1 presents an IoT architecture composed of edge, fog, and cloud. According to the OpenFog Consortium [2], cloud works as a coordination layer responsible for making general decisions (e.g., security, storage), putting almost all the processing, control, and security at the edge and fog layers. The core network (CN) connects the fog to the cloud. The fog layer is composed of powerful devices that should process and store data, and manage edge devices connected through the radio access network (RAN) (e.g., WLANs, cellular networks). The edge layer is composed of IoT end devices, which are also called IoT edge devices or just edge devices. They are usually resource-constrained devices and can preprocess data at the edge layer [4]. They use the RAN to send data to the upper layer.

The security challenges at the edge require a new security architecture for the devices [1]. Figure 2 presents a "security for IoT edge devices" taxonomy. It presents challenges to overcome, relevant attacks and how they can violate security requirements, and mechanisms that can strengthen security on IoT edge devices. The security challenges are [1–3]:

- *Simple and low-overhead building blocks*: Edge devices deal with scenarios involving life support and low-latency applications. Thus, security solutions should follow a simple and low-overhead architecture to meet applications' quality of service (QoS) requirements, which is a challenging task in resource-constrained devices.
- *Data preprocessing and decision making*: An edge device should be able to preprocess data from sensors and make decisions on it, which will draw the attackers' attention to perform new attacks.
- *IoT applications on the edge*: Most IoT applications are expected to migrate from the cloud to edge devices, which will make them the primary target of attacks.
- *Higher vulnerability to DoS attacks*: Devices can be a target for denial of service (DoS) attacks since attackers can more easily overwhelm them compared to cloud servers.

The challenges drive the rise of new threats at the edge. Also, old threats from traditional IoT environments are inherited by edge devices. Table 1 presents the most relevant attacks against them [3]. Also, it describes vulnerabilities and which security requirements can be violated in each attack.

IoT edge devices can be violated regarding unauthorized access, use, disruption, modification, or even destruction [4]. Most of the attacks aim to disrupt a device's software (e.g., operating system, virtualization, or key storage). However, vulnerabilities can also explore the network. For instance, attacks exploring backdoor security flaws have been seen in sensitive application environments like home care [3].

Regarding security violation, attacks explore the three pillars: confidentiality, integrity, and availability (CIA). Default passwords, for example, can be the reason for backdoor violations in edge devices (breaking confidentiality). Integrity can be violated when the attacker has control of the device, like in intrusion and physical/tampering attacks. Confidentiality and integrity can also be explored in privilege escalation attack. Finally, availability can be affected by DoS or physical/tampering attacks.

Based on this analysis and following the taxonomy (Fig. 2), we present the security mechanisms that should be used toward the definition of a security architecture for edge devices (more details are presented later) [1, 3, 5]:

- *Secure boot*: It is the basis for root of trust (RoT) systems. It requires a secure bootstrap subsystem in read-only memory or a hardware state machine to authenticate the stored program with asymmetric keys. If a modification is detected, the bootstrap process must be interrupted. A tamper-evident hardware module must protect the initial boot program.
- *Secure key storage*: A secure key storage system is based on specialized hardware to protect the integrity of keys. For instance, root public keys can be stored in a write-once memory to avoid key substitution. Alternatively, physical unclonable functions (PUFs) could be used not just for device authentication but also for runtime key generation, avoiding the necessity for key storage.
- *Security by separation*: This uses spatial and temporal separation to avoid software defects in a part of the device propagating to adjacent parts or the physical platform.

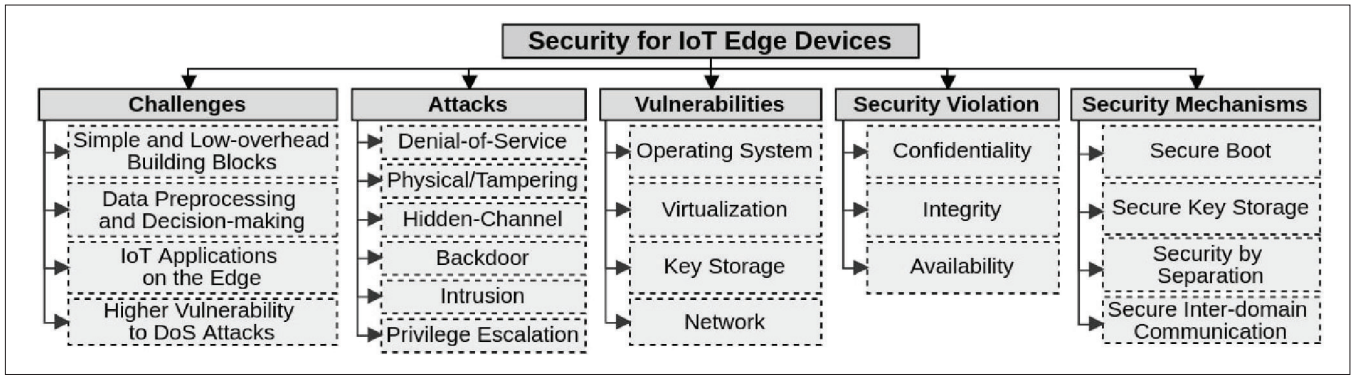

Figure 2. Security taxonomy for IoT edge devices.

- *Secure inter-domain communication*: There is a need for protection in the interaction between VMs and other software parts of an edge device.

The use of embedded virtualization in IoT is becoming predominant to strengthen the security around edge devices [2]. This trend focuses on better allocation through management of the available resources, while increasing the security of all elements of the edge systems related to both hardware and software [6].

Hypervisors are a crucial component of virtualized systems and represent an intermediate software layer between VMs and the actual hardware [6]. Running in a privileged context, a hypervisor has direct control of the hardware and can create and maintain trusted execution environments (TEEs) based on features such as isolation [5]. Also, it restricts lateral movement within the system while facilitating high-speed and secure inter-VM communications. Virtualization should be implemented with support from the hardware to be efficient and work with a small overhead and memory footprint.

In addition to virtualization, an authentication process should start from the hardware and pass through the virtualization layer until it reaches the applications running inside VMs. This process involves other important mechanisms, such as secure boot and secure key storage.

Attack	Vulnerability	Security violation
Denial of service	An attacker can spend resources of a device. It can be performed by stealing the device, manipulating its software, or disrupting the communication channel.	Availability
Physical/ tampering	Extraction of valuable cryptographic information for future use and software modification.	Confidentiality, integrity, availability
Hidden channel	Exploration of vulnerabilities regarding sharing of hardware components among the device's VMs. Data leakage across the VMs is a consequence of such attack.	Confidentiality
Backdoor	Remote access exploration. An attacker leverages backdoor programs through the network to break into the device's infrastructure without being discovered.	Confidentiality
Intrusion	An attacker takes control of certain sections of an edge device and can launch several types of attacks, such as DoS and selective information tampering.	Confidentiality, integrity
Privilege escalation	A malicious VM can manipulate other VMs or take control of certain elements of the device.	Confidentiality, integrity

Table 1. Attacks, vulnerabilities, and security violations against edge devices.

Security Architecture Definition

The proposed architecture is composed of four security mechanisms: secure boot, secure key storage, security by separation, and secure inter-domain communication. They ensure the authenticity of the executed code, the integrity of the runtime states, and the confidentiality of elements stored in the persistent memory. We present the proposed architecture in Fig. 3 and describe its mechanisms in the following subsections.

Chain of Trust Protection

A chain of trust (CoT) is established after various stages of secure boot, starting on the hardware and going to the highest level of software. An edge device must be designed to boot up only if the first piece of software to execute is cryptographically signed by a trusted entity (e.g., device vendor) and its signature matches a root public key that is stored in the device. Specialized hardware can be used to store cryptographic keys and perform the signature verification. The minimum requirements in terms of hardware are (Fig. 3A): a one-time-programmable (OTP) memory to store the root public key and a primary bootloader (i.e., a piece of software that runs before any operating system) embedded in the hardware or stored in protected memory that is able to verify the signature of the next boot stage. This scheme enables a RoT that should be used to ensure the integrity and authenticity of all running software. By anchoring this RoT in the hardware, tampering becomes more difficult. Once a RoT has been established, the initial software component should make identity and integrity checks with the hypervisor in the boot chain (Fig. 3B). If successful, the same process will take place in the next boot stage of the CoT until the software stack is fully protected [5].

Since a virtualized environment can be composed of VMs from different providers, an additional level of secure boot verification should be performed. The establishment of a CoT between the hypervisor and VMs is mandatory, allowing the hypervisor to be securely anchored to the hardware and trusted for all operations (Fig. 3C). For example, a vendor can mix third-party software (e.g., a robotic arm controller running in a virtualized application, with software developed in house). Virtualization makes the integration of different software's environments easier since it is

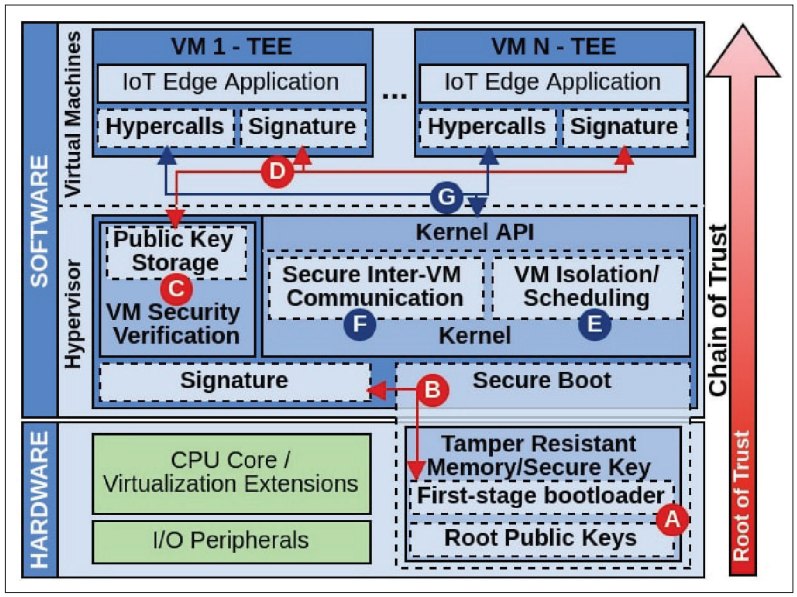

Figure 3. Proposed security architecture for IoT edge devices.

possible to keep them completely separated. Our security architecture allows the software providers to sign their VMs independently, that is, each VM has a signature header (Fig. 3D). The device vendor stores the provider's public key within the hypervisor, allowing it to check the integrity/authenticity of each VM individually.

While virtualization allows software stacks from different providers to be integrated, individually signed VMs ensure code integrity, authenticity, and non-repudiation. Thus, our architecture guarantees that the program has not been modified (integrity), the VM is from the provider it claims to be from (authenticity), and if a VM causes a malfunction, the responsible party cannot deny it (non-repudiation).

Virtualization Protection

Once the CoT is established, the system is still vulnerable to runtime attacks, and virtualization plays an important role in keeping the TEEs [1]. Additionally, the use of hardware-assisted virtualization, that is, specialized hardware with the capacity of simplifying the hypervisor implementation and improving the system's performance, contributes to reducing the attack surface.

Many different approaches are discussed about hardware assistance for hypervisors [7, 8]. As a result, most modern embedded hypervisors use a hybrid approach, applying full virtualization of the CPU (no modifications required in the virtualized software for basic functionalities), and paravirtualization (virtualized software must be modified) is required for extended services, such as inter-VM communication. Different hypervisor subsystems are used to improve devices' security, such as spatial isolation, temporal separation, and secure inter-VM communication.

The most common way of providing spatial isolation among VMs is through a memory management unit (MMU), a hardware block that provides virtual memory abstractions to the system (Fig. 3E). Processors with hardware-assisted virtualization implement a second stage of MMU translation, which is controlled by the hypervisor. Essentially, the VMs can handle the hardware in the same way as in a non-virtualized system. However, they map virtual memory to intermediate physical memory. The hypervisor is responsible for mapping intermediate physical memory addresses to physical memory, avoiding conflicts and ensuring separation between VMs. The second-stage MMU translation drastically decreases the hypervisor's exceptions, making it suitable for resource-constrained devices and with a small surface for attacks.

Temporal separation guarantees the correct distribution of processor time among VMs according to their execution priorities (Fig. 3E). Different authors have addressed the hypervisor's scheduler as a way to improve temporal separation and to honor real-time constraints [9]. Additionally, system interrupts require attention since they interfere directly with the VM's execution. Hardware-assisted virtualization can help to manage interrupts, allowing them to be redirected to VMs without intervention from the hypervisor. This feature is called interrupt pass-through, and it minimizes the overall hypervisor overhead and footprint.

Virtualized IoT edge applications require some level of interaction with each other and with the hypervisor itself. Also, some applications require secure channels for sensitive information. Thus, an efficient and secure inter-VM communication mechanism is available in the hypervisor (Fig. 3F). It is implemented as para-virtualized services, that is, using a well-defined hypercall application programming interface (API) (the VM's calls to the hypervisor) as presented in Fig. 3G. Thus, the hypervisor works as a communication arbiter, copying messages from the sender to the destination application. The hypervisor can check the size, the number of messages, and even deny forbidden communication.

Virtualization brings an advantage for keeping the integrity of TEEs: it allows the hypervisor to monitor the behavior of the VMs, detecting malfunction caused by software errors or attacks. There are three ways of detecting a compromised VM:

- A VM tries to access memory outside the address space defined at design time.
- A VM invokes hypercalls that should not be called.
- A VM does not periodically reset a watchdog.

If the system detects one of these situations, the VM will be restarted. Hence, two things can happen:

- If the VM code was compromised after deployment, its hash signature will not match during the chain of trust phase, and it will not boot up.
- If an attacker was exploiting a vulnerability based on the malfunction, it will boot up and run as expected. However, the hypervisor can emit alerts about the reset activities, enabling developers to investigate the causes.

Evaluation

We evaluated our architecture on a MIPS32 processor core running at 200 MHz, with 2 MB of flash memory and 512 kB SRAM. It is a resource-constrained device targeting IoT and embedded markets that supports hardware-assisted virtualization. We focused on exploring the

architecture's suitability for resource-constrained devices and security. Our evaluation consists of three different metrics: footprint, performance, and latency. Performance and latency results are based on an average of 1000 measurement runs. For security, we conduct a discussion around how the proposed architecture has achieved confidentiality, integrity, and availability.

Footprint, Performance, and Latency Analysis

We implemented the proposed architecture in about 10,000 lines of C code (LoC) for the hypervisor (including cryptographic algorithms) and 4000 LoC for the other security mechanisms. The implementations are publicly available as open source at https://github.com/hellfire-project. For the RoT, we implemented a secure boot (first-stage bootloader), storing it in the device's boot sector, which cannot be cloned to other devices. The secure boot mechanism implements digital signature using SHA256 for hash generation and two options for cryptographic algorithms: Elliptic Curve Digital Signature Algorithm (ECDSA) and RSA. On top of the secure boot, we implemented the virtualization support using our open source hypervisor, called Hellfire Hypervisor, which implements the same cryptographic algorithms as a secure boot mechanism. The Hellfire Hypervisor hosts entirely separated VMs that use the secure inter-VM communication mechanism to interact with each other. There are some advantages of being able to divide an IoT application to execute in smaller software components. First, each application is simpler and easier to implement and debug. Second, the only exposed application is the one that implements network communication thanks to the separation enforced by the hypervisor. Thus, potential attacks on this application will not expose sensitive data of other VMs or even allow the attacker to modify their contents.

Table 2a presents footprint results for each software component. SRAM presents the memory required to run each system. Code storage presents flash memory required to store the code. The total footprint (SRAM and code storage) including secure boot and hypervisor systems requires just 123 kB, which illustrates the small footprint of our architecture (applications are not considered in this measure). It is worth reiterating that the resulting footprint includes both cryptographic algorithms and the support for all virtualization features. For comparison purposes, off-the-shelf hypervisors like Xen and KVM [7] require tens of megabytes. Xvisor [8], a hypervisor designed for embedded systems, requires up to 16 MB of RAM. Some aspects help our hypervisor keep its small size: the paging subsystem is simplified (the number of VMs and their physical memory map are defined at design time), there is no filesystem implementation for devices with storage in flash, there is no support for an interactive shell or a proper filesystem, and all configuration is defined at design time. To improve the footprint analysis, we implemented two applications that execute on the bare metal, as there is no operating system running on top of the hypervisor. The IoT edge application is a simple monitoring application that receives data generated by the edge device itself and communicates with other VMs using hypercalls. It requires 32 kB of data SRAM for execution and 15 kB for code storage, resulting in a total of 47 kB. The network communication application is a more complex application that uses the network to communicate with other devices. It requires 64 kB of data SRAM for execution and 85 kB for code storage, resulting in a total of 149 kB. We used a small TCP/IP library, called picoTCP, to implement the network stack. If we consider both security architecture and applications, the total footprint required in this experiment is 319 kB, which is a promising result for resource-constrained devices.

(a)

Software	SRAM	Code storage	Total footprint
Secure boot	32	33	65
Hypervisor	32	26	58
IoT edge app	32	15	47
Network commun. app	64	85	149

(b)

VM size (kB)	SHA256 hash	Verification		Total time	
		ECDSA	RSA	ECDSA	RSA
32	11.15	57.10	39.40	68.25	50.55
64	23.20	57.30	39.40	80.50	62.60
128	46.25	57.70	39.40	103.95	85.65
256	92.65	57.50	39.40	150.15	132.05

Table 2. Results: a) footprint of software components (kB); b) performance of VM hash generation and signature verification (ms).

Table 2b presents the architecture performance for SHA256 hash generation from VMs stored in the flash memory and the ECDSA/RSA signature verification time of these VMs. For instance, a VM with a size of 64 kB takes 23.20 ms for hash generation, 57.30 ms for ECDSA verification, and 39.40 ms for RSA verification. Note that the hash generation time increases as the VM size gets bigger. On the other hand, time for signature verification is independent of the VM's size since it is based on the VM's hash, which is generated by the SHA256 algorithm and always has the same size. In these experiments, we used 3072 bits for RSA keys and 256 bits for ECDSA keys, which are equivalent in cryptography strength. Based on the results, ECDSA was more suitable for resource-constrained devices than RSA considering key length. However, RSA presented reduced execution time, which is also important in resource-constrained situations. Hence, the decision of the best algorithm depends on the requirements imposed by the application environment and the restrictions of resources in the device. Regarding latency, we observed that the communication latency between VMs is around 99 µs for messages up to 256 B, which outperforms related work, as discussed later.

Security Analysis

In this section, we evaluate the security of our architecture, showing how it has achieved the three fundamental elements of CIA: confidentiality (preventing sensitive device information from

Our model is based on embedded virtualization and trust mechanisms and ensures the security of IoT edge applications running on these devices. The design of our architecture does not require modification to IoT edge applications. As a result, the device's protection can be guaranteed without the requirement of re-engineering applications at the edge.

reaching the wrong people), integrity (avoiding improper device boot modification or destruction and ensuring its authenticity), and availability (ensuring reliable access to the IoT edge device).

In our system, an embedded application executes in a VM and takes advantage of two different security aspects: CoT and virtualization. The secure boot protection checks the software integrity at boot time. Thus, the main purpose of the CoT is to deliver a verified software stack to the runtime environment, helping to prevent physical/tampering attacks. The virtualization layer is responsible for preventing possible runtime violations. It keeps the attacker confined to the compromised VM, minimizing the severity of the attack and ensuring availability of other services. Also, the hypervisor can detect a VM's misbehavior without complicated intrusion techniques; for example, the call for a hypercall not predicted or message exchanges not expected. These detections may indicate that an attacker is in control of the VM, and in this case, the security mechanism properly intercepts such events.

The hypervisor's spatial separation, provided by the hardware MMU, improves confidentiality. If an application attempts to access a memory region of any other application or peripherals, it will be stopped by the hypervisor. Security by separation improves security over the following attacks (Table 1): DoS, backdoor, hidden-channel, intrusion, and privilege escalation. Additionally, the small hypervisor footprint, a result of its simplified subsystems, helps to keep a small attack surface. Recent research showed that two forms of attacks, named Meltdown and Spectre, allow for breaking the memory isolation, exposing sensitive data [10]. These attacks rely on out-of-order execution on modern processors. Our architecture prevents such attacks in two different ways:

- It avoids the use of affected processors since most of the embedded processors from MIPS and ARM families are not vulnerable.
- The CoT circumvents the execution of non-authorized software, a premise of the attacks.

Comparison with Existing Approaches

We compared our architecture with existing solutions regarding security mechanisms (Fig. 2) and highlighted their main differences. Pinto *et al.* [11] proposed a TrustZone-based architecture named IIoTTEED, which implements the basic building blocks of a TEE to protect edge devices. Spatial and temporal isolation mechanisms provide confidentiality and availability. Also, a secure boot process ensures integrity at boot time. The authors used a dual ARM Cortex-A9 running at 600 MHz for evaluation and concluded that IIoTEED must be complemented with other critical security strategies to guarantee tight industrial security for devices.

Jang *et al.* [12] proposed SeCRet, a framework that builds secure communications between a rich execution environment (REE) and a TEE. SeCReT creates a session key to sign the messages transferred during inter-domain communication. They evaluated SeCReT's performance on an Arndale board that offers a Cortex-A15 at 1.7 GHz in a dual-core processor. Results show that enabling SeCReT creates a performance overhead of 16.41 percent (from 1642.5 μs to 1912.1 μs) with an input payload of 256 B. In our approach, secure inter-VM communications happen through the use of hypercalls (99.59 μs for 256 B).

Dai *et al.* [13] present TEE, an architecture that uses the Xen hypervisor to allow multiple VMs on a commodity cloud-end platform to enjoy DRTM-like secure execution environments. However, according to Sabt *et al.* [5], Dynamic Root of Trust for Measurement (DRTM) is not suitable for low-overhead applications. They evaluated TEE's performance with an Intel Core Duo processor running at 1.8 GHz and 2 GB RAM. Results show that time to create the TEE domain is 173 ms with one vCPU and 64 MB memory, the TEE kernel is of 1.30 MB, and the time consumed for encryption is 436.9 ms (on average).

Guan *et al.* [14] present TrustShadow, a system that takes advantage of TrustZone technology to coordinate communications between applications and untrusted operating systems. It also implements secure boot and secure key storage mechanisms. The authors used an ARM Cortex-A9 processor, 1 GB DRAM and 256 kB iRAM for evaluation. The latency overhead on primitive operating system operations was 70 percent (on average).

While these systems have been shown to be effective in protecting IoT devices, they are over-architected for resource-constrained devices. The works present security solutions for more powerful devices, and it becomes clear when we compare the hardware they used to evaluate their works. On the other hand, our architecture is designed for resource-constrained devices and does not support some features, such as remote updates and terminal access, as other approaches do. However, our results outperform the existing approaches' results regarding footprint, latency, and cryptographic performance even using more resource-constrained devices than they did.

Conclusion

In this article, we have proposed a security architecture for IoT edge devices. Our model is based on embedded virtualization and trust mechanisms, and ensures the security of IoT edge applications running on these devices. The design of our architecture does not require modification to IoT edge applications. As a result, the device's protection can be guaranteed without the requirement of re-engineering applications at the edge. Also, the architecture presented promising results regarding memory footprint and inter-VM communications latency when compared to related work approaches. We expect that the proposed architecture can help in the design of security solutions for resource-constrained devices of upcoming edge computing architectures.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Finance Code 001.

References

[1] P. Zhang, M. Zhou, and G. Fortino, "Security and Trust Issues in Fog Computing: A Survey," *Future Generation Comp. Sys.*, vol. 88, 2018 , pp. 16–27.

[2] OpenFog Consortium, "OpenFog Reference Architecture for Fog Computing," Tech. Rep., 162 pp., Feb. 2017.

[3] R. Roman, J. Lopez, and M. Mambo, "Mobile Edge Computing: A Survey and Analysis of Security Threats and Challenges," *Future Generation Comp. Sys.*, vol. 78, 2018, pp. 680–98.

[4] A. Alrawais *et al.*, "Fog Computing for the Internet of Things: Security and Privacy Issues," *IEEE Internet. Comp.*, vol. 21, no. 2, pp. 34–42, Mar. 2017.
[5] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It Is, and What It Is Not," *14th Int'l. Conf. Trust, Security and Privacy in Computing and Commun.*, vol. 1, Aug. 2015, pp. 57–64.
[6] C. Moratelli *et al.*, "Embedded Virtualization for the Design of Secure IoT Applications," *27th Int'l. Symp. Rapid System Prototyping*, Oct. 2016, pp. 2–6.
[7] C. Dall and J. Nieh, "KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor," *19th Int'l. Conf. Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 333–48.
[8] A. Patel *et al.*, "Embedded Hypervisor Xvisor: A Comparative Analysis," *23rd Int'l. Conf. Parallel, Distributed and Network-Based Processing*, Mar. 2015, pp. 682–91.
[9] K. Cheng *et al.*, "Optimizing Soft Real-Time Scheduling Performance for Virtual Machines with SRT-Xen," *15th Int'l. Symp. Cluster, Cloud and Grid Computing*, May 2015, pp. 169–178.
[10] J. Horn *et al.*, "Meltdown and Spectre," Jan. 2018; https: // meltdownattack.com, accessed Oct. 25, 2018.
[11] S. Pinto *et al.*, "IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices," *IEEE Internet Comp.*, vol. 21, no. 1, Jan. 2017, pp. 40–47.
[12] J. S. Jang *et al.*, "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment," *Network and Distributed System Security Symp.*, Feb. 2015, pp. 1–15.
[13] W. Dai *et al.*, "TEE: A Virtual DRTM Based Execution Environment for Secure Cloud-End Computing," *Future Generation Comp. Sys.*, vol. 49, 2015, pp. 47–57.
[14] L. Guan *et al.*, "TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone," *15th Int'l. Conf. Mobile Systems, Applications, and Services*, June 2017, pp. 488–501.

Biographies

Ramão Tiago Tiburski (ramao.tiburski@acad.pucrs.br) received his M.S. degree in computer science from Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. He is a Ph.D. student of computer science at PUCRS. His research interests are IoT, fog and edge computing, and security for IoT resource-constrained devices.

Carlos Roberto Moratelli (carlos.moratelli@ufsc.br) received his Ph.D. in computer science from PUCRS. He is an adjunct professor at UFSC. He worked for 10 years in the telecommunication industry, acting on software engineering related to embedded systems. His research interests are embedded real-time systems, Linux Embedded, and virtualization for embedded systems.

Sérgio F. Johann (sergio.filho@pucrs.br) received his Ph.D. degree in computer science from PUCRS. He is an adjunct professor at PUCRS. He has experience in computer architecture design and organization, operating systems, embedded systems (design and integration), embedded software support, real-time systems and control systems.

Marcelo Veiga Neves (marcelo.neves@pucrs.br) received his Ph.D. degree in computer science from PUCRS. He is an adjunct professor at PUCRS. His primary research interests are software-defined networking, IoT and big data.

Everton de Matos (everton.matos@edu.pucrs.br) received his M.S. degree in computer science from PUCRS. He is an adjunct professor at Meridional Faculty (IMED). He is a Ph.D. student of computer science at PUCRS. His research interests are IoT, middleware, fog and edge computing, context awareness, and context sharing.

Leonardo Albernaz Amaral (lalbernaz@gmail.com) received his Ph.D. degree in computer science from PUCRS. He is adjunct professor at FTEC Faculdade de Tecnologia. He has experience in computer science with emphasis in middleware systems, RFID, IoT, smart cities, and pervasive systems.

Fabiano Hessel (fabiano.hessel@pucrs.br) is a full professor of computer science at PUCRS. He received his Ph.D. in computer science from UJF, France (2000). He has experience as General and Program Chair of several committees of prestigious conferences and journals. His research interests are embedded real-time systems, and RTOS and MPSoC systems applied to IoT/smart cities.