IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices

With the advent of the Internet of Things (IoT), security has emerged as a major design goal for smart connected devices. This explosion in connectivity created a larger attack surface area. Software-based approaches have been applied for security purposes; however, these methods must be extended with security-oriented technologies that promote hardware as the root of trust. The ARM TrustZone can enable trusted execution environments (TEEs), but existing solutions disregard real-time needs. Here, the authors demonstrate why TrustZone is becoming a reference technology for securing IoT edge devices, and how enhanced TEEs can help meet industrial IoT applications' real-time requirements.

Sandro Pinto, Tiago Gomes, Jorge Pereira, Jorge Cabral, and Adriano Tavares
University of Minho, Portugal

The Internet has changed the way we live, and the Internet of Things (IoT) is making the Internet even more immersive and pervasive. Nowadays IoT represents a collection of billions of smart, connected devices.[1] The ability to connect, manage, and control a device from anywhere and at any time leads IoT systems to generate, process, and exchange vast amounts of security-critical and privacy-sensitive data, turning them into attractive cyberattack targets.[2,3] Traditional protection mechanisms such as cryptographic algorithms and security protocols have proven inefficient,[3,4] because security is being misconstrued as the addition of features in a late stage of system development. The strong connectivity of IoT environments requires an holistic, end-to-end security approach, addressing security and privacy risks at all abstraction levels.[3]

Security by isolation is a well-established strategy for achieving security goals such as data confidentiality, integrity, and availability (CIA). Several software-based approaches such as microkernels, sandboxes, and virtualizations have been used,[5] but these methods fail in providing the desired security level. IoT systems must be extended with new security-oriented technologies that guarantee security from the outset. Among existing security-oriented technologies, TrustZone[6] is gaining particular attention due to the massive presence of ARM processors in the embedded market. By splitting the hardware and software resources into two worlds, TrustZone

 Published by the IEEE Computer Society 1089-7801/17/$33.00 © 2017 IEEE

Related Work in ARM TrustZone-Based Systems

Security, safety, and real-time processing are system-level requirements that drive the current development of Internet of Things (IoT) applications. There are several broad classes of approaches that have been applied to address them: software-based approaches for isolation such as microkernels, sandboxes, and virtualizations;[1] and hardware-based technologies such as trusted platform modules (TPMs), ARM TrustZone, and Intel SGX. Due to the extensive list of works on the security and real-time spectrum, we will focus our description on existing TrustZone-based solutions that individually or partially tackle the aforementioned requirements. To the best of the authors' knowledge there's no existing solution that fully and simultaneously addresses these requirements.

Peter Wilson and his colleagues pioneered the research around a trusted environment of typical ARM TrustZone systems.[2] They proposed an architecture that can be seen as a precursor for what the trusted execution environment (TEE) specifications standardized some years later. Johannes Winter developed a Linux-based virtualization framework to implement virtual TPMs using a software-only approach.[3] Marco Cereia and Ivan Cibrario Bertolotti implemented an asymmetric virtualization approach for real-time systems.[4] We implemented our own dual-OS solution[5] by running FreeRTOS side-by-side with Linux on a TrustZone-enabled Zynq platform. TZ-RKP[6] provides real-time protection of the OS kernel using the ARM TrustZone secure world. Implementing a non-bypassable event-driven monitoring strategy, TZ-RKP uses memory protection to prevent attacks that aim at modifying an OS kernel running on the non-secure world side. SeCReT[7] builds a secure communication channel between the rich execution environment (REE) and TEE because no message-protection mechanism exists in TrustZone. By creating a session key to sign the messages transferred during interdomain communication, the processes in the REE can communicate with TrustZone securely. Johannes Winter's research group developed the ANDIX OS[8] to evaluate the applicability of ARM TrustZone to secure future industrial control systems, and later compared such an approach against a security controller.[9]

References

1. F. Armand and M. Gien, "A Practical Look at Micro-Kernels and Virtual Machine Monitors," *Proc. 6th IEEE Consumer Communications and Networking Conf.*, 2009, pp. 1–7.
2. P. Wilson et al., "Implementing Embedded Security on Dual-Virtual-CPU Systems," *IEEE Design & Test of Computers*, vol. 24, no. 6, 2007, pp. 582–591.
3. J. Winter, "Trusted Computing Building Blocks for Embedded Linux-Based ARM TrustZone Platforms," *Proc. 3rd Workshop on Scalable Trusted Computing*, 2008, pp. 21–30.
4. M. Cereia and I.C. Bertolotti, "Virtual Machines for Distributed Real-Time Systems," *Computer Standards & Interfaces*, vol. 31, no. 1, 2009, pp. 30–39.
5. S. Pinto et al., "Towards a Lightweight Embedded Virtualization Architecture Exploiting ARM TrustZone," *Proc. IEEE Int'l Conf. Emerging Technologies and Factory Automation*, 2014, pp. 1–4.
6. A.M. Azab et al., "Hypervision Across Worlds: Real-Time Kernel Protection from the ARM TrustZone Secure World," *Proc. ACM Conf. Computer and Comm. Security*, 2014, pp. 90–102.
7. J. Jang et al., "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment," *Proc. Network and Distributed System Security Symp.* (NDSS), 2015; www.internetsociety.org/doc/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment.
8. A. Fitzek et al., "The ANDIX Research OS: ARM TrustZone Meets Industrial Control Systems Security," *Proc. 13th IEEE Int'l Conf. Industrial Informatics*, 2015, pp. 88–93.
9. C. Lesjak, D. Hein, and J. Winter, "Hardware-Security Technologies for Industrial IoT: TrustZone and Security Controller," *Proc. 41st IEEE Ann. Conf. Industrial Electronics Soc.*, 2015, pp. 002589–002595.

enables a secure environment to coexist with a rich execution environment (REE), while ensuring that sensitive data are completely isolated and protected from unauthorized accesses. Several academic and commercial trusted execution environment (TEE) solutions have been proposed, but they lack in providing the real-time requirements of Industrial IoT (IIoT) applications.[3]

IIoT applications such as industrial control systems are increasingly looking for consolidation of several critical and noncritical functions into a single device to reduce equipment and factory footprint and consequently the overall cost.[3] Moreover, with the advent of smart services,[3,7] equipment needs to send and receive status information, control commands, and configuration updates from its manufacturer. In this sense, IIoT controllers not only need to guarantee functionality isolation and real-time requirements of control algorithms, but also need to protect their integrity against unauthorized modification, as well as ensure the veracity of their inputs. While in the context of industrial control systems, the notion of security has traditionally almost the same meaning as safety (that is, protection of human lives and machines against system failures), with integration of information technology, protection against cyberattacks became a major design goal.

In this article, we demonstrate how the Industrial IoT Trusted Execution Environment for Edge Devices (IIoTEED) can meet the real-time and

security requirements of IIoT edge devices, dictated by the three elements of CIA. We propose a TrustZone-based architecture that implements the basic building blocks of a TEE as a lower-priority thread of a real-time operating system (RTOS). The RTOS was slightly modified to support trusted applications (TAs) and to schedule the REE only during the idle periods. Experiments demonstrate security is assured while the system's real-time properties remain nearly intact. Our main contributions are

- design and implementation of a new TrustZone-based architecture that implements an enhanced TEE that meets the IIoT requirements;
- evaluation of the implemented solution to measure the impact on the real-time properties of the system while it leverages the three fundamental elements of CIA; and
- discussion about how IIoTEED will perfectly align with resource-constrained edge devices, and why it must be complemented with other critical security strategies.

For others' work in this area, see the related sidebar.

ARM TrustZone

TrustZone technology[6] refers to security extensions implemented by ARM since the ARMv6 architecture. These extensions provide a secure and separate execution environment that protects the integrity and confidentiality of secure–sensitive processing, by splitting the hardware and software resources into two worlds — the secure and the non-secure world.

TrustZone Hardware

The most significant architectural innovation of the TrustZone hardware architecture is the addition of a new 33rd processor bit, the non-secure bit, which indicates in which world the processor is currently executing. To switch between the secure and the non-secure world, a special new secure processor mode, called monitor mode, was introduced. To enter the monitor mode, a new privileged instruction was also specified — secure monitor call (SMC). The TrustZone address space controller (TZASC) extends security at the memory level, by enabling partition of DRAM into different memory regions. The TrustZone-aware memory management unit (MMU) provides two distinct MMU interfaces, and the isolation is still available at the cache-level. System devices can be dynamically configured as secure or non-secure through the TrustZone protection controller (TZPC). The generic interrupt controller provides several interrupt models, allowing the configuration of fast interrupt requests (FIQs) and interrupt requests (IRQs) to secure or non-secure interrupt sources.

TrustZone API

The TrustZone API (TZAPI) is an application API that specifies how non-secure applications (NSAs), running on the rich environment, interact with the isolated execution environment. Following a client–server model, the API defines a set of abstract software interfaces by which an NSA can interact with a TA. The API allows clients to send commands and requests to a TA, and exchange data between both worlds. Secondary API features include querying the properties of installed applications as well as downloading new security TAs at runtime. The TrustZone API doesn't include any specification about how to develop applications running inside the isolated execution environment.

IIoTEED

The TEE is a secure area ensuring that sensitive data are stored, processed, and protected in an isolated and trusted environment. Typical TrustZone-based TEE solutions embody a small secure kernel, responsible for managing TAs, on the secure world side, and a rich OS, responsible for managing NSA, on the non-secure world side. The secure OS is only scheduled explicitly under request of the rich environment, when an NSA needs to access sensitive data. While this approach perfectly fits in several application domains where real-time performance isn't a concern (for example, mobile phones), it's not well-suited for a multitude of domains where real-time processing is a key requirement (for example, IIoT).

Figure 1 depicts IIoTEED, our proposed TrustZone-based architecture that provides a safe and secure environment, completely isolated from the REE, protecting the integrity and confidentiality of secure–sensitive processing while enhancing availability by isolating critical processing from the noncritical one. Security-related operations and real-time processing are performed on the secure world side, while the general-purpose and rich environment lies on the non-secure side. The software running in the secure world

is composed by the Monitor layer, the Trusted RTOS (T-RTOS), and its corresponding real-time tasks and TAs. The monitor component, running in monitor mode, works as a gatekeeper and is responsible for preserving the state of each world during the world switch operation (that is, the change from the secure to the non-secure world side and vice versa). The T-RTOS, running in kernel mode, corresponds to a lightweight version of an RTOS extended with some security capabilities, which allows not only the execution of real-time tasks but also TAs. The software running in the non-secure world side consists of a general-purpose operating system (GPOS) with the respective TZAPI-dependent software (that is, the privileged TrustZone kernel module and the unprivileged TZAPI library) and the NSA.

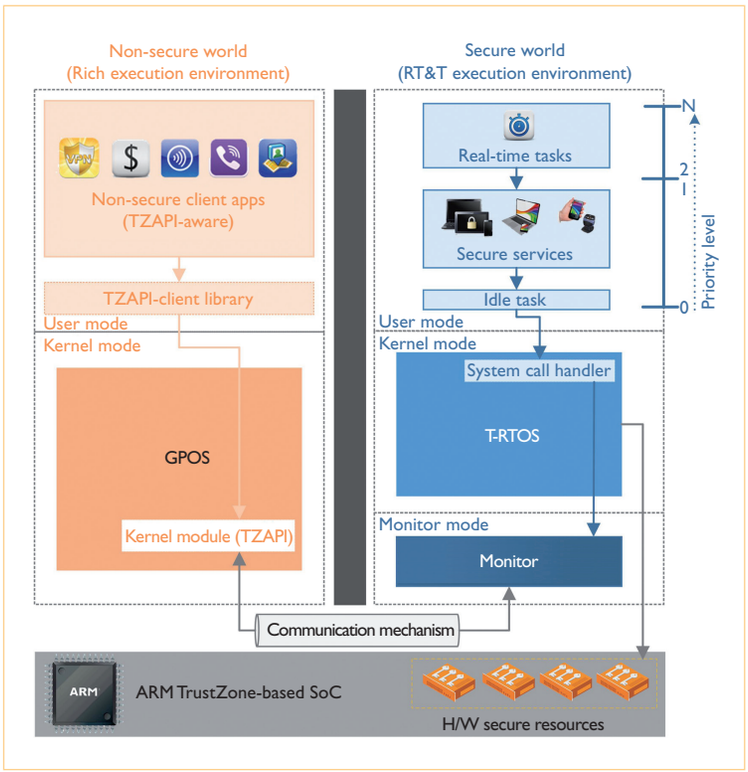

Figure 1. Proposed Industrial IoT Trusted Execution Environment for Edge Devices (IIoTEED) architecture. GPOS = general-purpose operating system; H/W = hardware; RT&T = real-time and trusted; T-RTOS = Trusted real-time operating system; TZAPI = TrustZone API.

Secure World Components

The secure world software provides a lightweight, real-time environment extended with some security capabilities. The monitor is responsible for every world entry and exit, as well as for establishing a communication channel between both worlds. The T-RTOS is responsible for guaranteeing the real-time characteristics of the system and simultaneously interacting with NSA during its idle periods.

T-RTOS is a modified version of an RTOS extended with some security capabilities. The main modifications on the kernel side encompass the implementation of the idle scheduling policy, the addition of some system calls, and the support for the TZAPI communication. First, to preserve the system's real-time properties, the idle task was modified to implement the idle scheduling policy: the RTOS has a greater scheduling priority than the GPOS, and consequently the GPOS is only scheduled during RTOS idle periods. To support the addition of new security features, some system calls were implemented: because the idle task and secure services run in user mode, requesting new kernel services dictates the addition of specific system calls. Last, a new small kernel module for managing the TZAPI communication was implemented; it's responsible for interpreting the commands/data received from the NSA and acting accordingly to the desired operation. If the requested action has to be handled by the TA, the request is forwarded to the user space.

The monitor component, although running at a higher privileged level (monitor mode) than the T-RTOS, is configured to behave in a passive way so that the T-RTOS has the processor as long as real-time tasks are ready-to-run. Hence, from the secure side, the monitor will be dispatched only when T-RTOS is idle or returning from a TA by invoking specific system calls that will trigger an SMC instruction. On the other hand, once the GPOS starts executing, the monitor will be invoked through the specific SMC instruction (that is, through TrustZone kernel module) as well as later when an FIQ is triggered (for example, T-RTOS systick). In any case, the monitor will perform a world switch operation by saving the state of the current world, and restoring the state of the ready-to-run world. Due to the intrinsic TrustZone hardware capabilities in banking an extensive list of processor and coprocessor registers, the world control block (that is, data structure with the state of each world) is minimal and composed of only 28 registers.

Non-Secure World Components

The non-secure world software provides the foundation for application developers to design

and implement standard NSAs that interact with TAs. The GPOS provides a rich and flexible environment by which NSAs, following the TZAPI specification (TZAPI library), interact with TAs through the TZAPI kernel module.

The TZAPI-client library exposes the standardized API defined by TrustZone specification, abstracting the application developer from the specificities of the TrustZone message formats and the TrustZone kernel module input/output control (IOCTL) calls. We implemented the complete specification, which includes the descriptors, control functions, encoder and decoder functions, service manager functions, and asynchronous operations, with the exception of functions related to the runtime download and removal of services.

The TrustZone kernel module implemented for the GPOS (Linux) provides a pseudo-character device that implements a logical communication channel (between the non-secure and the secure world) on top of the real communication channel, and provides the functional foundation to implement the non-secure world TZAPI library. It provides a set of specific IOCTLs that semantically understands parameters, allocates memory buffers, encodes and decodes data, prepares the requests, and establishes the communication.

Communication

IIoTEED implements a remote procedure call (RPC)-style communication interface to establish a communication channel between the NSA and TA. RPCs are always initiated in the non-secure world side, where NSAs use the available system call interfaces (kernel module) to explicitly invoke the SMC instruction. This SMC instruction will trap the execution flow into the monitor mode, where the monitor component is responsible for restoring T-RTOS execution as well as forwarding allocated message buffers information that passed through the core registers. Once the T-RTOS is restored, the idle task is recovered and the TA dispatcher will forward the incoming request to the respective TA. The communication follows a blocking-implementation strategy, which means the non-secure world side will only be recovered on completion of the RPC request. Once it happens, the TA notifies the T-RTOS, which in turn goes through the SMC handler and returns to the last well-known execution point of the non-secure world side.

Evaluation

Our solution was evaluated on a ZedBoard targeting a dual ARM Cortex-A9 running at 600 MHz. Despite using a multicore hardware architecture, our current implementation only supports a single-core configuration. We focused our evaluation on real-time processing (experiment 5.1) and security (experiment 5.2). To evaluate the system's real-time properties we targeted three metrics: performance, determinism, and interrupt latency. To evaluate the security, we conduct a discussion around how IIoTEED has achieved confidentiality, integrity, and availability.

Real-Time Processing

To measure the impact on real-time system properties, we split our experiments in two parts. To assess the performance and determinism metrics, we compared the native single-core version of the FreeRTOS (v. 7.0.2) against T-FreeRTOS, using the Thread-Metric Benchmark Suite. To ascertain the system interrupt latency, we performed specific microbenchmarks. MMU, caches, branch predictor, and others dynamic architectural features were disabled in the secure world side. We used a performance monitoring unit to assess the world switch and latency overhead.

The Thread-Metric Benchmark Suite consists of a set of specific benchmarks to evaluate RTOS performance. The suite comprises seven benchmarks, evaluating the most common RTOS services and interrupt processing. Each benchmark outputs a counter value, representing the RTOS impact on the running application: the higher the value, the smaller the impact. We collected 50 samples for each benchmark, corresponding to a total of 700 collected samples for both test case scenarios. Assessed results demonstrate the overhead introduced by our approach, when compared to the single execution of FreeRTOS, is negligible (<0.0001 percent). Regarding determinism, the assessed variance was in the same order of magnitude in both test scenarios. This is perfectly understandable because once T-FreeRTOS starts running real-time tasks, it will never be interrupted by any security-related feature. Furthermore, all introduced kernel modifications were carefully implemented to first privilege the execution of real-time features. For example, in conditional statements (such as "if" or "switch"), secure features were introduced below real-time related statements, just to avoid compromising the execution flow.

Interrupt latency is the measurement of the system's response time to an interrupt, which corresponds to the elapsed time between interrupt assertion and the instant a response occurs. Equation 1 expresses the system latency where τ_H is the hardware dependent time that depends on the interrupt controller on the board as well as the type of the interrupt, τ_{OS} is the OS-specific induced overhead, and τ_{WS} is the monitor-specific induced overhead (world switch):

$$\tau_{IL} = \tau_H + \tau_{OS} + (\tau_{WS}). \quad (1)$$

Our experiments showed that latency in the native system (FreeRTOS) is 0.89 microseconds, which corresponds also to the average interrupt handling overhead of our system. The last part of Equation 1 represents the extra overhead induced by our approach, but which only happens when the RTOS has no real-time ready-to-run tasks, and consequently the monitor is invoked to perform a world switch. Because the monitor runs with all interrupt sources disabled, the worst case scenario happens when an FIQ request (for example, RTOS tick) arrives while a context switch from the secure to the non-secure world is starting. In this case, the request is handled only after two complete world switches, which corresponds to a worst-case interrupt latency of 8.11 microseconds. This is a sporadic situation that happens under rare conditions, because two asynchronous and independent events need to occur at the same time: an asynchronous FIQ needs to be triggered while a world switch is happening. Nevertheless, because the overhead introduced on latency has a deterministic upper bound, it can be taken into account when designing the real-time system.

Security

In this section, we evaluate the security of our solution by summarizing how IIoTEED has fully or partially achieved the three fundamental elements of CIA: confidentiality, integrity, and availability.

Confidentiality. This feature is the ability to restrict data to only those with authorized access. IIoTEED partially provides confidentiality by means of TrustZone' strong spatial isolation mechanisms. The GPOS can't access any memory segment allocated to the T-RTOS, because the TZASC traps any unauthorized memory access. The GPOS with a separated MMU and cache interface nullifies any cached information leakage. Moreover, it can't access any device assigned to the T-RTOS, because the TZPC also traps any unauthorized device access. The only possible access path is through the communication channel, where there exists one well-known security breach of TrustZone.[8] The current design of TrustZone's architecture doesn't authenticate access to resources, enabling man-in-the-middle attacks and so, interception and manipulation of messages transferred through the channel. Side-channel attacks are also out of scope of the ARM TrustZone specification.

Integrity. This element enforces the consistency, accuracy, and trustworthiness of data and of the system over its entire life cycle. IIoTEED provides integrity only at boot time, through the secure boot process. Once the system is booted, TrustZone per se doesn't provide any hardware mechanisms to assure the integrity of data over time. A software-based solution for introspection (for example, a health monitor) could be implemented, however we believe hardware trust anchors such as security controllers will fit better in the IIoT domain. A hybrid approach using TrustZone and security controllers, as envisioned by Johannes Winter's research group,[7] will assure continuous checking of component authenticity as well as data and system integrity to prevent manipulation.

Availability. This characteristic emphasizes that authorized parties are able to access the information when needed. T-RTOS proved to have a high-level of availability, guaranteed by the strong temporal isolation (asymmetric scheduling policy) as well as by the coexistence of privileged (FIQs) and unprivileged (IRQs) interrupt sources. By scheduling the GPOS only on T-RTOS idle periods, as well as preempting its execution once an FIQ is triggered, we were able to guarantee a high-level of availability at the secure world side. Our experiments focused on performing some tests and attacks to the Linux system running on the non-secure side, and observing how they could disturb the correct behavior of the T-RTOS. The first experiment forced several Linux reboots. We have observed that the nonexistence of services from the GPOS while rebooting doesn't affect any service type provided by the T-RTOS. Then, we injected a device driver on Linux to

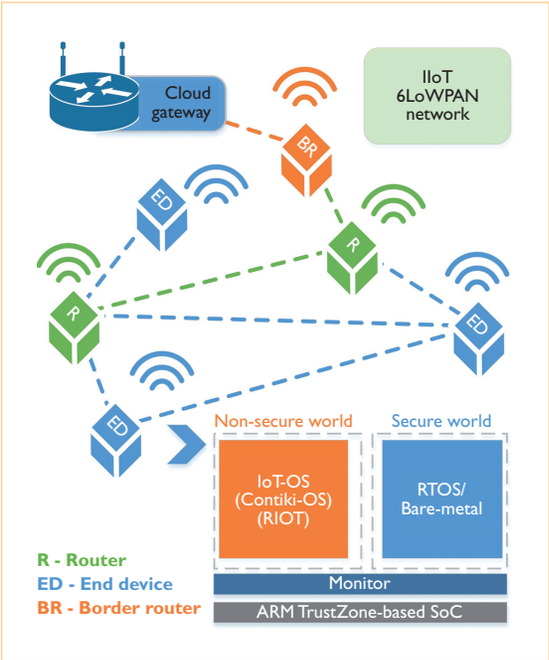

Figure 2. Future Industrial IoT (IIoT) application scenario. 6LoWPAN = IPv6 and low-power wireless personal area networks; SoC = system on chip.

reconfigure the MMU interface of the non-secure world side to try to access a memory area outside the boundary of the non-secure memory area. Due to the existence of one MMU interface for each world, as well as the strong memory isolation provided by the TZASC, the attempt was completely unsuccessful. Finally, we connected a radio transceiver to the system, linked and managed by Linux, that's able to receive data packets from several sensors on a sensor hub. We've tested the behavior of a compromised sensor by repeatedly sending data bursts to our edge device, which repeatedly generated interrupt requests on Linux. This experiment simulates a denial-of-service attack to the system. Due to the coexistence of privileged (FIQs) and unprivileged (IRQs) interrupt sources, FIQs belonging to the T-RTOS were able to preempt the execution of Linux, even when executing an IRQ request.

Envisioned Future of IIoTEED

IIoTEED, in its current implementation, can act as a suitable solution for high-end edge devices, but in a smart industry environment the supremacy of resource-constrained devices demands a similar solution. The recent ARM's decision about introducing TrustZone technology to the new Cortex-M and Cortex-R processors series opens the possibility to widen our IIoTEED solution to the resource-constrained edge devices with minimum engineering effort. Industrial applications with real-time requirements, for example, data acquisition or permanent monitoring systems, will benefit much more from the given technology, allowing resource-constrained edge devices to connect to the network while real-time execution and security are never jeopardized.

Figure 2 partially depicts a possible simplified and generic IIoT architecture. A large number of topologies within an infinite number of configurations and requirements will have to coexist on the same environment — such as constrained edge devices with monitoring applications, router devices capable of forwarding wireless packets over the network, and border router devices responsible for interfacing the IIoT network to the Internet. According to the device's role in the network and the corresponding architecture, the IIoTEED environment will be able to perform in different configurations. For example, for a middle-end edge device, IIoTEED can be configured to run an IoT-based OS on the non-secure world side (such as Contiki-OS or RIOT), and an RTOS on the secure world side. On a low-end edge device, IIoTEED can be configured to run a lightweight IoT-based OS on the non-secure world side, and a bare-metal support for TAs on the secure world side. Complex network operations that usually demand higher processing capabilities — securing the communication channels by encrypting data using strong security, for example — are also covered as the new processors include hardware peripherals to perform these tasks.

Currently, IIoTEED only partially addresses the industrial security puzzle at edge level. To help with end-to-end security while meeting the right tradeoff between security and real-time requirements, it must be extended with other tailored, hardware-assisted security and acceleration strategies, mainly those implementing edge device identities as well as measures to continuously monitor and protect the authenticity of data exchanges and firmware upgrades. Edge device authentication not only prevents counterfeit spare parts and repair tools, but it also reinforces edge device encryption and transport layer security.

The IoT is an emerging key technology that paves the way for the next generation of smart industrial systems. In today's IoT systems,

the functional and security requirements aren't totally fulfilled. The strong connectivity of IoT environments require an end-to-end security approach, addressing security at both the device, network, and cloud levels. ARM TrustZone addresses security at the device level, and is being used as a foundation for a TEE realization. However, the TEE specification doesn't address the real-time needs of industrial applications. IIoTEED offers an enhanced TEE for IIoT edge devices. Experiments demonstrated real-time properties of the system that remain practically unaffected while security is guaranteed from the outset. However, to guarantee tight industrial security, IIoTEED must be complemented with other critical security strategies for edge devices (for example, hardware-assisted device identity) to better leverage the three fundamental elements of CIA.

Current research aims at GlobalPlatform standards' implementation. With interoperability and standardization as the top of our goals, we've already implemented a huge part of the GlobalPlatform TEE client and TEE internal specifications. Our future research roadmap will focus on implementing our vision about the applicability of this solution for resource-constrained edge devices, as well as reinforcing their security by integrating other hardware trust anchors.

Acknowledgments

This work has been supported by COMPETE (POCI-01-0145-FEDER-007043) and Fundação para a Ciência e Tecnologia (FCT) under grant SFRH/BD/91530/2012 and UID/CEC/00319/2013.

References

1. L.D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Trans. Industrial Informatics*, vol. 10, no. 4, 2014, pp. 2233–2243.
2. S.L. Keoh, S.S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet of Things J.*, vol. 1, no. 3, 2014, pp. 265–275.
3. A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and Privacy Challenges in Industrial Internet of Things," *Proc. 52nd Design Automation Conf.*, 2015, pp. 54:1–54:6.
4. R. Langner, "Stuxnet: Dissecting a Cyberwarfare Weapon," *IEEE Security & Privacy*, vol. 9, no. 3, 2011, pp. 49–51.
5. F. Armand and M. Gien, "A Practical Look at Micro-Kernels and Virtual Machine Monitors," *Proc. 6th IEEE Consumer Comm. and Networking Conf.*, 2009, pp. 1–7.
6. J. Winter, "Trusted Computing Building Blocks for Embedded Linux-Based ARM TrustZone Platforms," *Proc. 3rd Workshop on Scalable Trusted Computing*, 2008, pp. 21–30.
7. C. Lesjak, D. Hein, and J. Winter, "Hardware-Security Technologies for Industrial IoT: TrustZone and Security Controller," *Proc. 41st IEEE Ann. Conf. Industrial Electronics Soc.*, 2015, pp. 002589–002595.
8. J. Jang et al., "SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment," *Proc. Network and Distributed System Security Symp.*, 2015; www.internetsociety.org/doc/secret-secure-channel-between-rich-execution-environment-and-trusted-execution-environment.

Sandro Pinto is a researcher and a PhD student at the Embedded Systems Research Group at Centro Algoritmi, University of Minho. His research interests include operating systems, virtualization, and security for embedded, cyber-physical, and IoT-based systems. Pinto has an MS in electronics engineering from the University of Minho. Contact him at sandro.pinto@dei.uminho.pt.

Tiago Gomes is a researcher and a PhD student at the Embedded Systems Research Group at Centro Algoritmi, University of Minho. His research interests include embedded systems hardware/software codesign for resource constrained wireless devices, wireless protocols for low-rate wireless personal area networks and, network protocols for the IoT. Gomes has an MS in telecommunications engineering from the University of Minho. Contact him at mr.gomes@dei.uminho.pt.

Jorge Pereira is a researcher and a PhD student at the Embedded Systems Research Group at Centro Algoritmi, University of Minho, Portugal. His research interests include operating systems, hardware/software codesign, virtualization, and security technologies. Pereira has an MS in industrial electronics and computers engineering from the University of Minho, Portugal. Contact him at jorge.m.pereira@algoritmi.uminho.pt.

Jorge Cabral is an assistant professor with the University of Minho, Portugal. His research interests include embedded systems applications. Cabral has a PhD in microsystems technology from Imperial College London, UK. Contact him at jcabral@dei.uminho.pt.

Adriano Tavares is an associate professor at the University of Minho, Portugal, and a visiting professor at Jilin University, China. His research interests include embedded, cyber-physical, and IoT-based systems' modeling and design; system software design; system-on-chip design; and engineering education. Tavares has a PhD in industrial electronics from the University of Minho. Contact him at atavares@dei.uminho.pt.