Secure Channels in an Integrated MPSoC Architecture

Haris Isakovic, Armin Wasicek
Institute of Computer Engineering
Vienna University of Technology
Email: {haris.isakovic, armin.wasicek}@tuwien.ac.at

***Abstract*—Providing security in an embedded system often boils down to solving a trade-off problem between security and performance. Simultaneously, Multi-Processor System-on-a-Chip (MPSoC) devices are in the early stages to increase computational performance, energy and die area efficiency, and reduce the number of physical units in the embedded system design arena. Moreover, MPSoCs enable composing heterogeneous subsystems on a single silicon die which is particularly desirable for large volume embedded devices. However, these benefits come at a price: an increase in the system's complexity. Complexity does not only make the system design process more difficult, but also it renders certain vulnerabilities possible. A solution is to follow well-established architectural principles to reduce complexity and to provide the required level of security. In this paper we demonstrate how the basic architectural principles of the ACROSS MPSoC architecture can be combined with the requirements of standard security techniques (i.e., encryption, authentication) to produce an efficient security solution for MPSoC systems. We propose a security architecture which uses the principles of temporal and spatial partitioning, temporal determinism, and mixed-criticality integration to migrate resource expensive security functions form the application components to a dedicated security component within the MPSoC. This leaves application components with a thin security provider, without any loss of functionality and more local resources at their disposal. Thereby, we deliver a flexible, resource efficient security solution, which highlights the benefits of partitioning MPSoC architectures for security.**

I. Introduction

A security architecture describes how the security design artefacts are positioned and invoked, and how they relate to the overall system architecture. A security architecture's goal is to establish the non-functional properties confidentiality, integrity, and availability for authorised users [1]. Technically, the portion of the system that sustains these three properties is called the Trusted Computing Base (TCB). A TCB [2] is a *small amount of software and hardware that security depends on and that we distinguish from a much larger amount that can misbehave without affecting security*. Hence, defining what is inside and what is outside a system's TCB is the critical security design task.

Integrated systems implement a variety of different types of computational units. They are a very challenging category for system designers, because each subsystem can have completely different properties. Multi–Processor System–on–a–Chip (MPSoC) platforms are currently emerging as a very powerful representatives of heterogeneous systems. The flexibility of an MPSoC with heterogenous components can facilitate distributing the application's computational load between components. This can be a particular benefit in embedded system applications, which often require a special mix of processing tasks.

However, these benefits of heterogenous systems come at a price. According to the European Aviation Safety Agency (EASA) [3], most existing MPSoCs are classified as 'highly complex micro controllers'. With respect to security, complexity is a major source of vulnerabilities [4]. Therefore, a MPSoC architecture should provide complexity reduction techniques not only for system design purposes, but also to facilitate a more secure system [5]. The most common engineering approach to reduce complexity is *divide et impera* – partition the system in controllable subsystems. Then, the problem to be solved by the architecture is how to join the subsystems in a secure way.

Another challenge on the road to a secure system is the tradeoff between performance and security. Mechanisms used to enforce security are usually resource demanding. Particularly, in resource-constrained environment like embedded systems this tradeoff is a critical point for security design. In safety-critical systems, resource allocation is often guided by over-dimensioning in order to provide enough resources at a critical instant. This design approach might lead to an expensive design solution.

In this paper, we propose a possible security architecture based on the ACROSS MPSoC [6]. The ACROSS MPSoC facilitates a solution for the design and implementation of hard real-time applications. It represents a new generation of MPSoCs with integrated Time-triggered Network–on–a–Chip (TTNoC). The ACROSS MPSoC provides several non-functional properties (i.e., temporal determinism, temporal isolation, spatial isolation, partitioning). The main contribution of this paper is to show, how these properties can be used in combination with standard security mechanisms to improve the organisation and implementation of a generic security architecture.

The remainder of the paper is organized as follows: Section II focuses on partitioning architectures, Section III provides overview of security architecture basics, Section IV elaborates how partitioning is used to design security, in Section V we present our findings, and Section VI concludes the paper.

II. Partitioning Architectures

In this section we describe the concept of partitioning in computer systems. Our main focus is on partitioning in the

embedded systems domain and MPSoC platforms.

Kopetz [7] illuminates three major design strategies to counteract complexity for the embedded system designer: partitioning, abstraction, isolation and segmentation. Numerous examples of partitioning can be found in biology (i.e., ant colonies [8]), as well as in system design. Partitioning a highly complex computer systems into smaller independent functional units is one of the most frequent techniques to master complexity. A common definition of a partition is *'a system partition or logical partition is a subset of computer's hardware resources, virtualized as a separate computer'* [9]. This definition mostly refers to software partitions implemented through a virtualization, but there are other ways to implement partitioning on a computer system. If we observe partitions as independent functional units with certain properties, than each sub-system of a larger system which possesses these properties can be called logical or system partition. In practice, a partition or a functional unit is realized as a software/hardware artefact that works in isolation, but is connected via well-specified interfaces to the other parts of the system. Following this definition it is apparent that a partition's properties manifest in its interfaces. Thus, we can deduct all required and existent properties of a partition by analyzing its interfaces [10]. Clearly, the concept of a partition is very well applicable in the security domain, because a security architecture first partitions a system in different entities and then defines access rights between these entities.

Two properties of a partition are very relevant for the design of embedded real-time systems:

- *Spatial Isolation*. A spatially isolated partition owns a part of memory which cannot be accessed by any other partition. The guarantee of the spatial isolation increases confidentiality and integrity of the partition's private data. These two properties are also part of the security specification of a partition and the system in general.
- *Temporal Isolation*. A temporally isolated partition has a guaranteed uninterruptible execution time by any other partition. This means that temporal characteristics of that partition cannot be influenced in any way. This property increases execution determinism, it can be used to predict a system behaviour. Therefore it supports dependability and reliability of a partition, which are preconditioning properties for safety.

In order to apply the principle of partition on a subsystem, at least one of these properties must be achieved. Other important properties are: energy consumption, computational power, security, fault propagation. The partitions are independent units and if they are spatially and temporally isolated their additional properties mutually incremental. The properties of an individual partition add up to the same property on the system level.

In order to analyse a computer system, we want to apply the concept of partitioning on two levels:

- *Software Partitions.* The partitioning mechanisms are realized in software, for instance, in the operating system.
- *Hardware Partitions* are implemented by dividing a system's physical resources (i.e., processors, memories, communication controllers) into functionally independent units.

In the next two sections we describe the methods and techniques used in the state-of-the-art partitioned systems, to achieve partitioning. We show how these different types of the partitions stack up, in the systems with mixed partition types (i.e., MPSoC).

A. Operating System Level

On the Operating System (OS) level, techniques to achieving partitioning are often summarised under the hood of virtualization. Virtualization has been introduced by IBM in 1972 for the mainframe computers [11]. Since then virtualization became important for every computing domain, from powerful internet servers to small embedded systems. There is a large spectrum of relevant techniques, ranging from simple process virtualization to virtualizing an entire computer system.

A virtualization manager is the part of the system that spawns the partitions. It is usually called Hypervisor or Virtual Machine Monitor (VMM). A Hypervisor can be implemented as a process that emulates an OS. Therefore, it can contain again different processes which might translate to execution threads of the original OS hosting the Hypervisor. This technique is called Paravirtualization. In this case, the Hypervisor encapsulates an application in a process. Consequently, the strength of the isolation between encapsulated applications depends on how the original OS manages the resources between different processes. Other virtualization techniques are full virtualization [12] and Hardware Assist, which relies on special hardware modules which support the virtualization. The full virtualization uses binary translation technique and direct execution techniques. This means that Hypervisor takes the instruction set used by the Guest OS and translates it to the one executed on the underlying hardware.

Another way to achieve software partitioning are microkernels. Contrarily to virtualization whose origins are in the mainframe computers, microkernels were soon adopted by the embedded computing community. A microkernel provides only the most basic operations needed to implement an OS. Other functionalities (e.g., a networking stack) are shifted in so-called servers which run as processes in the user space. Microkernels are an important architecture type for Real-Time Operating Systems (RTOSs). In the embedded domain important representatives of the microkernel-based OSs are INTEGRITY [13], LynxOS [14], OKL4 [15] and PikeOS [16].

Another way to achieve software partitioning is using a microkernel. A microkernel provides only the most basic operations needed to implement an OS. Other functionalities (e.g., a networking stack) are shifted in so-called servers which run as processes in the user space. Microkernels are an important architecture type for RTOSs.

In the embedded domain some of the microkernel-based OSs which implement partitioning are INTEGRITY [13], LynxOS [14], OKL4 [15] and PikeOS [16]. T

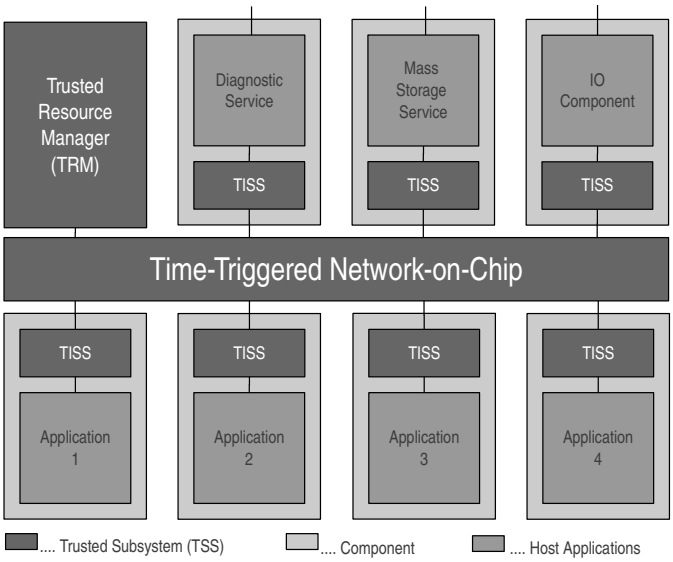

Fig. 1. Schematic diagram of ACROSS MPSoC: Application (1,2,3,4) and system components (Diagnostics, Mass Storage, IO) are composed via the Time-Triggered NoC

B. MPSoC Level

The future of embedded systems is in MPSoCs. Microcontrollers, Field–Programmable Gate Arrays (FPGAs) and other common computing elements will be pooled on a single silicon die shaping a heterogeneous System–on–a–Chip (SoC) for energy efficiency, size, cost, and integration reasons. These elements will be connected by a dedicated communication infrastructure like point-to-point (P2P) channels, a bus, or Network–on–a–Chips (NoCs) [17]. MPSoCs are a powerful solution, but apart from the benefits like versatility and performance there are some drawbacks like complexity. For instance, using an MPSoC in a safety-critical applications is currently not advisable, because the certification issues.

As a research initiative, the ACROSS MPSoC (see Figure 1) is set out to simplify certification for safety-critical applications in different application domains. This is achieved by implementing partitioning and encapsulation. Opposed to some virtualization techniques that use hardware features, the ACROSS MPSoC is totally designed to segregate its components in partitions in hardware. A component is a container for any system or application functionality. Communication between partitions is done through so-called encapsulated communication channels that are defined over a Time-Triggered NoC [18]. End-points of these encapsulated communication channels are called *ports*. The generic interface used for message exchange between SoC components and the TTNoC is called Trusted Interface Subsystem (TISS). Messages exchanged over the TTNoC are strictly bound to their temporal specification. Furthermore, memories are not shared, each partition is required to have its own. Components are containers for functionalities. Application components realize the required functions for the application whereas system components are responsible for correct operation of the MPSoC's native functions (i.e., IO, Mass Storage, Synchronization, etc).

At first, the ACROSS MPSoC was designed to host hard real-time systems. But the applied design principles have a strong impact on the implementation of a security architecture. Using the powerful concept of partitioning on the hardware level, we are able to imitate physically distinct units of computation. A similar design can be found in current industrial-grade SoC solutions like the ARM TrustZone (secure and non-secure mode of operation) [19] and the Cell processor's 'Secure Vaults' [20].

C. Combination

Hardware and software partitioning mechanisms can be combined to facilitate a two-level partitioning approach. In ACROSS, a component can instantiate a partitioning OS like for instance PikeOS. This results in a fine grained partitioning architecture. Each software partition spawned inherits the properties given through the execution on a specific hardware.

In our case, a PikeOS partition will execute on a component of the ACROSS MPSoC. Therefore, it has some very special properties when communicating with other partitions on the same MPSoC. Although integrated on the same SoC two instances of PikeOS cannot invalidate temporal or spatial guarantees of each other. From the application's view the usage of a partition is always the same. Partitions communicate through so-called queuing and sampling ports, be it that they are hosted on the same component or on different components. This gives a designer a the freedom to split the application in as many partitions as needed and distribute the partitions on components as wanted. When communicating with partitions on the same components, a partition has the usual properties that are given by PikeOS. When connecting to an external device, a partition acts as a physically distinct entity.

In the next section we present a generic security architecture for communication that we will put on top of the partitioning concept of the ACROSS MPSoC in the subsequent section.

III. Secure Communication Architecture

A security architecture is the part of the overall system architecture, which describes how does the system satisfy security requirements [21]. In this section we describe one of the most common design pattern of any security architecture, a *Secure Channel*. Associated with a Secure Channel are the Secure Kernel and a key management facility.

Figure 2 depicts the big picture of the secure communication architecture. It depicts the architectural elements that refine the TISS which is the standard interface to access the TTNoC. On the bottom, a regular communication through a standard channel without any security guarantees is given. If a Secure Channel should be used, the send and receive operations are executed on the Secure Provider which transparently provides a channel with security properties to the application. All Secure Channels of a component are administered and maintained by the Secure Kernel.

A. Secure Kernel

The Secure Kernel is considered to be the TCB in an ACROSS MPSoC. The security of a system cannot be compromised if the TCB is operating according to specification. The Secure Kernel is responsible to manage keys and access

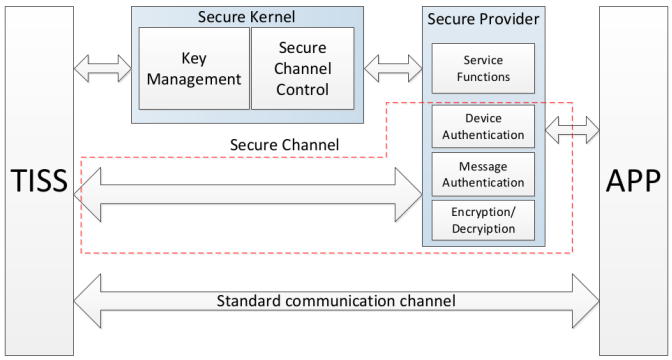

Fig. 2. Block diagram of Secure Communication Architecture

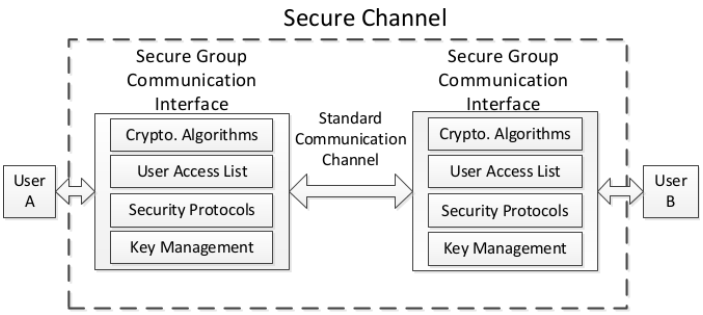

Fig. 3. Secure Channel Structure

rights of all Secure Channels of a component. Therefore, an application does not have to take care of any bookkeeping, but it can invoke the communication API for secure and regular channels in the same way.

The Secure Kernel is an independent thread which executes several tasks important for establishing of secure channels. It executes in its own partition and is therefore isolated from the application's functions. Its memory cannot be accessed from the application partitions and it embodies only the minimal amount of confidential information. Hence, even if it would be compromised, the effect would concern only the active connections of the compromised component. An attacker could not use it to gather keys from other components.

When the channel is opened from an application a key for that channel is requested from the Secure Kernel. This request contains the port ID of the channel, which is then used as an input for the minimal perfect hash function. The hash function returns the index of the corresponding key in the key table managed by the Secure Kernel. If keys have to be exchanged with some other entity, the Secure Kernel performs this action in a secure and reliable manner.

B. Secure Channel

The Secure Channel is an abstract object composed of a regular communication channel and a security protocol implemented on the communication channel. A protocol is defined as a series of steps, which involve two or more parties with a goal of achieving certain task. The security protocol uses cryptographic algorithms to solve a security related issue on a communication channel. Which security mechanisms are actually used in a security protocol is defined by requirements of an application. This can be implemented differently from application to application.

The basic structure of a secure channel is shown in Figure 3 and comprises following elements:

- Cryptographic algorithms which provide basic cryptographic services
- Security protocols which extend the capabilities of the cryptographic algorithms and provide security properties to communication channels
- A user access list to manage access rights to the channels
- Key management system provides secure storage and distribution of the keys used for various security protocols and cryptographic algorithms.

Establishing a Secure Channel requires coordination and connection of several basic services of the ACROSS service model. The secure channels in ACROSS are unidirectional single-cast channels based on symmetric encryption security mechanisms. Here the procedure that instantiates a Secure Channel in the ACROSS MPSoC:

1) Secure Channels are created by the application using the related API
2) The API evokes the Secure Channel Provider which opens a special storage holding unit for the channel. Thus, Secure Channels are host by a Provider and the Secure Kernel handles security key management.
3) Secure Channel Provider opens a single port on the TTNoC for each channel, which is then used to transmit data for the channel. The port configuration data is static and shared among channel among parties before compile.
4) A secure channel are used only for the off-chip communication, therefore a gateway (IO component) is instructed to establish communication with the other device. This requires configuring ports on both sides and the gateway to relay the messages.

C. Key Management

The key management defines the process of a key generation, key storage and a key distribution. Keys are generated either during the pre-configuration phase or during runtime. Each channel is provided with the single key which are managed by the Secure Kernels. The Key management exists once per physical device. Hence, it is implemented as a dedicated system component. Access is facilitated only via the rigorously specified interface at the TTNoC.

IV. Using Partitions to Design for Security

The security design problem we are solving in this section is how to map the general security architecture defined in the previous section to partitions. In our system model, partitions are established through the physical separation between cores as well as the separation mechanisms of the OS running on each core. Generally spoken, it is assumed that hardware security mechanisms are stronger than software mechanisms, because hardware is naturally not as modifiable as software.

Basically, we reason about partitions that execute on the same core and partitions that run on different cores. Partitions interact only via their communication (linking) interfaces. Communication channels are statically allocated before runtime. The integrity of the MPSoC's configuration is continuously checked by the Trusted Resource Manager (TRM). Then an encapsulated communication channel can be:

- *Local*: the parties are located on the same component and communicate via an OS mechanism like a message queue
- *On-chip*: the parties reside on different cores of the same MPSoC and they are communicating only using the TTNoC
- *Off-chip*: the parties reside on different MPSoCs and for the communication they use first the TTNoC, then a gateway to an off-chip network (e.g., TTEthernet)

A. Fine-grained control of encryption

Depending on the mode entities communicate Secure Channels are instantiated and the strength of their security mechanisms is determined. Embedded systems often do not dispose over bountiful resources and must compensate on some properties to achieve optimal performance. Each of the above described modes has a different impact on the realization of a Secure Channel. The concrete impact is determined by the attacker model used. For instance, we want to assume that an attacker can tap, eavesdrop, and manipulate the device's pins and their physical connections. It is out of scope for the attacker to open the device's package and read out the TTNoC[1] Furthermore, we also assume that reading out a component's memories (e.g., via microprobing) is not feasible.

Using such an attack model, we can determine which Secure Channels actually requires security processing and which considered to be safe. This has to be determined for each Secure Channel individually. Therefore, our architecture also accounts for instantiations with channels with mixed security requirements (i.e.,). Giving the designer such a fine-grained control over the execution of security protocols and hence cryptographic algorithms facilitates the design principle *Economy of Mechanism*. Moreover, the implementation of the security architecture can be more performant and less resource intense. This is particularly important for Embedded Systems, because they have often stringent resource constraints regarding computational power, memory sizes, and energy budget.

B. Offloading intense computations

The second direct benefit of our proposed generic security architecture is that computationally intense operations can be securely offloaded to system components. For instance, the key management which runs as system component in a separate hardware partition performs the creation of new cryptographic keys. Key creation can be a highly demanding process. At least, it involves computing a random number generator and conversion of random bits in the appropriate key format. In case of asymmetric encryption, primality testing and operations on big integer numbers are required in addition.

[1]If instantiated as an ASiC, it is easily to visually locate the connections of the NoC in the layout. It would have as well have a larger feature size than e.g., the devices implementing a processor.

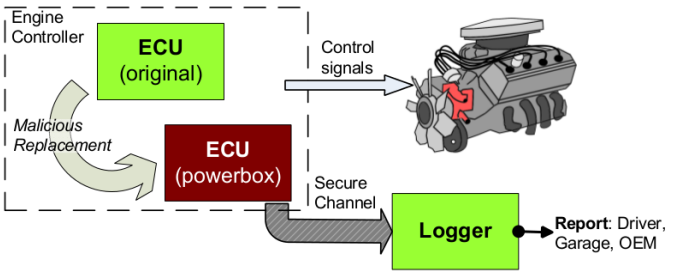

Fig. 4. Powerboxing: The original ECU is replaced by a powerbox that contains different (more powerful) control algorithms for a specific engine. In our experimental setup, a logger is connected via a Secure Channel to detect such a manipulation.

The key management has to solve a producer-consumer problem. It has to produce enough cryptographic keys that are then consumed by the respective Secure Kernels. Traditionally, key creation is implemented as a background task on the same computer that perform send/receive operations. This has a major drawbacks: considering a real-time system, this background task might not receive enough resources in a critical instant and the application could run out of cryptographic material. When integrating several subsystems into a single device like aspired by the ACROSS MPSoC, the benefits of a shared key management component is getting obvious: The components can focus on what they actually want to do, namely sending and receiving messages. Key creation can be managed by a specialised component. This has also an impact on the energy consumption of the device. If no background tasks are running, power gating is trivial. Moreover, if the key management component has a dedicated hardware support, its operations can be perform energy-efficiently.

V. Use Case: Power Boxing

As a use case, we use a Secure Channel to monitor values of an automotive Engine Control Unit (ECU) to detect malicious manipulations in the control system software, for instance through a power boxing attack [22].

Powerboxing [23] directly modifies the output signals of an ECU by inserting a hardware module in the vehicle (see Figure 4). The rationale behind this idea is similar to chip tuning: an attacker wants to tap the engine's maximum potential. The inserted module either replaces the original ECU on the communication system or it is placed as a man-in-the-middle between the original ECU and its connected actuators. The installation in the vehicle is fairly simple as demonstrated by several supplier guidelines and demonstration videos on the internet. It requires plugging the new unit to the power supply and to reconnect the network cables from the original ECU to the powerboxing module. Example is a fuel injector tuning chip which changes signals between ECU and a fuel injector module. It increases power of the engine by increasing the injection time of the injector. The principal block diagram is shown in Figure 4.

Such manipulation can be effectively detected by continuously monitoring an ECU's output. We used a Secure Channel

to periodically output an authenticated message containing the current state of the engine controller. The controller is implemented in an ACROSS MPSoC. The receiving device logs the state and its authenticity. Absence of the correct authentication indicates external manipulation of the signals. The logger might then report to the driver, a garage, or even the Original Equipment Manufacturer (OEM) to inform that the current setup has been compromised.

The experimental setup for the original ECU was done by our colleagues from the control department partly in simulation and partly with real devices. In a subsequent step, we added the Secure Channel to the demonstrator as a proof-of-concept for our security architecture. Moreover, we showed that the operation of the Secure Channel does not interfere with the temporal guarantees given by the component, hence, it is minimally invasive.

VI. Conclusion

Partitioning is a valuable concept to effectively reduce complexity in system design. In this paper, we showed how partitioning can successfully applied to design and implement a secure communication architecture. Starting with the abstract notion of a secure channel we broke down the required security protocols and cryptographic algorithms in different partitions where the respective functionalities are implemented. This straight-forward design approach is enabled by the strong isolation and encapsulation guarantees of the underlying system architecture. In our case be build on an MPSoC architecture intended for integrated, hard real-time systems. Layering the presented security architecture on top does not interfere with established timeliness and dependability properties as we demonstrated in the automotive case study. Wrapping up, we showed that implementing security is feasible even in strictly constrained environments like hard real-time systems, if a sound architectural approach is followed. Clean separation of functionalities is enabled by implementing partitioning as a foundation.

Acknowledgment

This document is based on the ACROSS project in the framework of the ARTEMIS programme. The work has been funded in part by the ARTEMIS JU and National Funding Agencies of Austria, Germany, Italy and France under the funding ID ARTEMIS-2009-1-100208. This research was in part supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme under the funding ID PIOF-GA-2012-326604. The responsibility for the content rests with the authors.

References

[1] O. S. Architecture. It security architecture. [Online]. Available: http://www.opensecurityarchitecture.org/cms/en/definitions/it-security-architecture

[2] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 265–310, 1992.

[3] E. A. S. Agency. Development assurance of airborne electronic hardwa re. [Online]. Available: http://www.easa.europa.eu

[4] G. McGraw, *Software Security*. Addison Wesley, 2006.

[5] A. Wasicek and C. E. Salloum, "A system-on-a-chip platform for mixed-criticality applications," in *Proceedings of 13th IEEE International Symposium on Object/component/service-oriented Real-time distributed computing (ISORC)*, May. 2010.

[6] C. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek, "The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, Sept., pp. 105–113.

[7] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[8] M. Albrecht and N. Gotelli, "Spatial and temporal niche partitioning in grassland ants," *Oecologia*, vol. 126, pp. 134–141, 2001. [Online]. Available: http://dx.doi.org/10.1007/s004420000494

[9] K. Singh. Security on the ibm mainframe. [Online]. Available: http://www.redbooks.ibm.com/redbooks/pdfs/sg247803.pdf

[10] H. Kopetz and N. Suri, "Compositional design of rt systems: A conceptual basis for specification of linking interfaces," in *Research report, Technische Universitt Wien, Institut fuer Technische Informatik, Treitlstr. 1-3/182-1, 1040*, 2003, pp. 51–60.

[11] I. Corp. z/vm^{TM} built on ibm virtualization technology. [Online]. Available: http://www.vm.ibm.com/pubs/HCSF8A50.PDF

[12] vmware. Understanding full virtualization, paravirtualization, and hardware assist. [Online]. Available: http://www.vmware.com

[13] G. H. Software. Integrity rtos. [Online]. Available: http://www.ghs.com/products/rtos/integrity.html

[14] LynuxWorks. Lynxos-178 rtos - lynuxworks. [Online]. Available: http://www.lynuxworks.com/rtos/rtos-178.php

[15] O. K. Labs. Okl4 microvisor. [Online]. Available: www.ok-labs.com/products/okl4-microvisor

[16] S. AG. Pikeos rtos and virtualizations. [Online]. Available: http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/

[17] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 23:1–23:20, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1255456.1255460

[18] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *RTCSA*, 2008, pp. 120–129.

[19] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," *ARM white paper*, 2004.

[20] K. Shimizu, H. P. Hofstee, and J. Liberty, "Cell broadband engine processor vault security architecture," *IBM Journal of Research and Development*, vol. 51, no. 5, pp. 521–528, 2007.

[21] M. Gasser, *Building a secure computer system*. New York, NY, USA: Van Nostrand Reinhold Co., 1988.

[22] A. Wasicek, "Copy protection for automotive electronic control units using authenticity heartbeat signals," in *10th IEEE International Conference on Industrial Informatics (INDIN)*, 2012, pp. 821–826.

[23] H. Prankl and H. Schaufler, *Motortuning zur Leistungssteigerung an Traktoren: Bericht ; Projekt-Nr. BLT 05 3322*. FJ BLT, 2006. [Online]. Available: http://books.google.at/books?id=fto-MgAACAAJ