# The Security of ARM TrustZone in a FPGA-Based SoC

E. M. Benhani, L. Bossuet, *Senior Member, IEEE*, and A. Aubert

**Abstract**—Cybersecurity of embedded systems has become a major challenge for the development of the Internet of Things, of Cloud computing and other trendy applications without devoting a significant part of the design budget to industrial players. Technologies like TrustZone, provided by ARM, support a Trusted Execution Environment (TEE) software architecture and are inexpensive integrated solutions. While this technology allows isolation and secure execution of critical software applications (e.g., banking), recent preliminary works highlighted some security breaches or limitations when the ARM processors are embedded in a FPGA-based heterogeneous SoCs such as the Xilinx Zynq or Intel SoC FPGA devices. This paper highlights the security issue of such complex SoCs and details six efficient attacks on the ARM TrustZone extension in the SoC. A prototype system design on a Xilinx Zynq SoC is the target of the attacks presented in this paper but they could be adapted to other SoCs. This paper also includes recommendations and security solutions to design a trustworthy embedded system with a FPGA-based heterogeneous SoC.

**Index Terms**—FPGA Security, embedded system design, ARM TrustZone

## 1 Introduction

The design of embedded systems requires the implementation of software and hardware components in the same integrated devices because of the many constraints. At the beginning of 2000s, some FPGA companies anticipated these advances in design by integrating microprocessor cores in FPGA fabrics. One and a half decades later, most FPGA companies offer FPGA-based SoCs, which are stronger than ARM processors and FPGA fabric combined. The association allows designers of embedded systems to develop powerful and efficient software-hardware systems that benefit from software flexibility and hardware performance. Moreover, ARM processors with embedded security features are capable of reliably executing sensitive software applications. This ARM security system is known as TrustZone technology [1].

TrustZone is a hardware security extension of ARM processors that helps divide both hardware and software resources into two worlds, one secure and one non-secure. Each world has its own private resources and shares others with the second world. The TrustZone-enabled processor has a monitor mode that controls the interaction between the two worlds. With such an embedded security system, the FPGA-based SoCs can be used to design a trusted embedded system that provides a trusted execution environment (TEE) [2]. This is a major advantage because FPGA-based SoCs are used in increasing number of sensitive applications with security requirements [3]. For example, FPGA-based SoCs are used to secure data processing on the cloud [4], [5], to implement internet protocol security (IPSec) [6], [7], to secure communication in software radio systems [8], to secure cyber physical infrastructure [9], industrial control systems and safety-critical systems [10], [11].

However, the security of TrustZone technology in the entire SoC, including the FPGA fabric, requires careful analysis. Indeed, the FPGA party embeds IP from third parties that may be malicious and several internal side channels can be exploited to access confidential data. So designers need to understand possible attack scenarios and their benefits.

The contribution of this paper is to highlight the security breaches of modern FPGA-based heterogeneous SoCs that exploit ARM TrustZone technology. A first step in evaluating the TrustZone in modern FPGA-based heterogeneous SoCs was discussed in [12]. The present paper describes six efficient attacks on the ARM TrustZone extension in a FPGA-based heterogeneous SoC. The paper also discusses protection solutions in the literature, proposes new solutions and makes design recommendations.

The paper is organized as follows. Section 2 reviews the state of the art and highlights possible threats to ARM TrustZone technology and FPGA-based heterogeneous SoCs. Sections 3 and 4 provide background on ARM TrustZone technology and on the AMBA AXI bus required to fully understand the attacks described in the paper. Section 5 presents the system prototype, based on a Xilinx Zynq-7010 device that was used for the experiments presented in Section 6. Section 7 describes and recommends new design methodology and security protection and design. Section 8 concludes the paper.

## 2 State of the Art

Studying the security of TrustZone technology in a FPGA-base SoC requires exploring works that address TrustZone

• *E. M. Benhani, L. Bossuet and A. Aubert are with the Hubert Curien Laboratory, University of Lyon, Saint-Etienne 42000, France.*
*E-mail: {elmehdi.benhani, lilian.bossuet, alain.aubert}@univ-st-etienne.fr.*

*Manuscript received 25 June 2018; revised 18 Jan. 2019; accepted 13 Feb. 2019. Date of publication 17 Feb. 2019; date of current version 17 July 2019.*
*(Corresponding author: L. Bossuet.)*
*Recommended for acceptance by P. R. Schaumont.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TC.2019.2900235*

security, FPGA security and FPGA-based SoC security. We review the state of the art in these different fields.

## 2.1 TrustZone Security

As a processing system executing sensitive applications, TrustZone is threatened by many attacks from the system to the hardware level.

At the system level, Roseberg [13] studied a vulnerability affecting Qualcomm implementation of the TEE (QSEE). The vulnerability was present in a wide range of android mobile devices supporting TrustZone and using a Qualcomm Snapdragon SoC. Roseberg succeeded in executing a non-secure code in a secure world by using a failure of handling integer overflows in Security Monitor Call (SMC) request function. This exploit is possible even in the presence of a security patch by software downgrading [14] or privilege escalation using a test feature like JTAG [15].

At the system level, it is possible to cause a fault during execution to bypass protection or to perform a cryptanalysis. Lipp [16] showed that the rowhammer effect from a non-secure world can be used to attack a TrustZone secure world. This was confirmed by Carru [17], who described a successful attack to recover a private key stored in the secure memory of an RSA signature implementation, by making a specific read of a row in the dynamic random-access memory (DRAM) accessible. With the bit flip caused by the rowhammer effect, the attack bypassed the TrustZone system that normally prevents software executed in a non-secure world from writing to secure memory (only accessible from the secure world). Tang et al. [18] used Dynamic Voltage and Frequency Scaling (DVFS) to induce a fault in the seventh round of an AES encryption executed in the secure world. They used a malicious kernel driver installed in the non-secure world to attack the AES executed in the secure world. The malicious kernel driver pushed the operating frequency of the device beyond the allowed margins, while the AES was implementing loop encryption. The device used in their attacks was a mobile device with a TrustZone-enabled SoC. DVFS can also be used as a covert channel to secretly send information from one thread to another, as shown in [19]. In the same way, a thermal channel can be used as a covert channel [20].

At the system level, side channels such as cache-based attacks are also efficient against the TrustZone. To perform a cache-based attack, the cache memory has to be shared between several processors or applications. Zhang et al. [21] presented a study on cache timing-based information leakage from the ARM TrustZone. Using the Prime and Probe cache attack, they succeeded in recovering the full key of an AES software implementation based on t-table. In this attack, the attacker fills the cache with known states in the non-secure world before executing the cryptographic operation in the secure world and observes the changes in the state of the caches. These authors explained that the leakage was due to a fundamental design choice of the TrustZone-enabled cache architecture, which aims to improve system performance by allowing the two worlds to share the same cache hardware. Other examples of cache-based attacks on ARM are provided in [22], [23], [24].

At the hardware level, physical side channel analysis is also possible. Bukasa et al. [25] proved that the TrustZone does not affect the efficiency of the electromagnetic analysis. They succeeded in attacking an AES encryption using Correlation Power Analysis (CPA) on a TrustZone-enabled device. They also succeeded in attacking a verified PIN algorithm using template attacks. Their two attacks underlined the risks faced by users who use a SoC for critical operations with no protection against physical attacks. The efficiency of electromagnetic analysis was also demonstrated in [26] and [27].

Another hardware level threat is the fault injection attack. Majeric et al. performed an electromagnetic injection locally on the ARM cryptographic accelerator and assurance module (CAAM) during AES computations [27]. The CAAM was physically located by electromagnetic analysis before the fault injection attack. This is a typical scenario of physical cryptanalysis of an embedded cipher, but fault injection can also be used to target other sensitive parts of the device such as secure storage. Rivière et al. suggest using electromagnetic injection to target the control flow, especially the instruction cache, of an ARMv7-M architecture [28]. Their study proves that such an attack can replace and/or skip instructions and can break countermeasures or bypass the security process. In the same way, [29] and [30] suggest using power supply glitch and laser beam injection to attack the secure boot by avoiding the security process (loaded code authentication).

## 2.2 FPGA Intrinsic Security

As hardware systems, FPGAs are sensitive to all physical attacks (side channel analysis, fault attacks) [31], [32]. Additionally, FPGAs are concerned by specific threats to their configuration [31], [32], [33]. New kinds of threats recently appeared concerning the transfer of sensitive data from one part of the FPGA fabric (one IP) to another, or outside the device.

Schellenberg et al. showed how to use the power distribution network of a FPGA to perform a side channel analysis of an AES cipher internally [34]. To this end, they used a delay sensor based on a buffer chain with a stable clock as input. Indeed, the path delay of the buffer chain depends on the supply voltage and can be measured internally. Ring oscillators can also be used thanks to their sensitivity to variations in the supply voltage. In [35], the authors studied this solution to prove that information leaks along the long wire of the FPGA fabric and can be intercepted internally by a malicious IP. To send the sensitive information discreetly from the FPGA to outside the device, it is possible to use a ring oscillator as a very low cost transmitter when electromagnetic emission is being considered for the communication channel, as proposed in [36].

## 2.3 FPGA-Based SoC Security

To the best of our knowledge, only two works have specifically addressed the security of FPGA-based heterogeneous SoC embedded ARM TrustZone technology. First, Jacob et al. showed how a malicious IP can access processor core features and memory to bypass software or system security such as the secure boot [37]. This paper demonstrates how a secure boot is compromised by using a malicious IP despite the use of cryptographic functions for boot code and bitstream verification. Nevertheless, the ARM TrustZone feature can counter the attack on the secure boot coming from the FPGA. Indeed, if the TrustZone is configured before the FPGA is programmed, it is possible to declare all transactions arriving via slave ports of the processing system (typically memory access) as non-secure. In this way, even if the master IP (on the FPGA fabric) declares itself to be secure using the AXI protocol, the transaction remains non-secure and the master cannot access the secure memory. Second, a first step in evaluating the security of the ARM TrustZone technology in an FPGA-based SoC was discussed in [12]. That paper was the first to present modifications in malicious hardware made to take advantage of security failures in the FPGA fabric to bypass ARM core security. Here, we extend that pioneer work by providing details of the attack scenarios and by showing how the FPGA-based SoC security should be addressed by taking all the design aspects into account, from the CAD tool to the insertion of third party IP.

This review of the state of the art clearly identified threats related to FPGA based-SoC, but other threats are possible because FPGAs design was never security centric. This paper describes several security breaches of FPGA-based SoCs that should encourage FPGA suppliers to propose a secure-by-design FPGA-based SoC.

The rest of the paper highlights the security issues of the propagation of the TrustZone outside the ARM processor in a heterogeneous SoC. The two following sections present some technical aspects of the ARM TrustZone technology and AXI bus.

## 3 Elements of ARM TrustZone Technology

The TrustZone technology is an ARM hardware security extension built in a microprocessor core. It partitions both hardware and software resources into two worlds, one secure and one non-secure. The secure world protects high-value code and data while the non-secure world executes a rich operating system environment. ARM TrustZone is easy to use [38] to protect private data from any untrustworthy operating system or malicious software and to develop a trusted execution environment (TEE) [39].

From the point of view of hardware, the TrustZone is a set of IP blocks that allows the application designer to partition the memory, the I/O peripherals, the interruptions, and software execution into secure and non-secure worlds. Two internal bits of the system architecture, the *NS* bits (called *AWPROT* for write transactions and *ARPROT* for read transactions), inform the full system that the executed software is run in the secure world (the logic state of the *NS* bits is low) or the non-secure world (the logic state of the *NS* bits is high). The *NS* bits are two extra control bits on the main system bus called AMBA3 AXI bus (see the following section for more details).

The seven main hardware blocks of the TrustZone are:

1. The TrustZone protection controller (*TZPC*): controls the partitioning of the peripherals into secure and non-secure worlds.
2. The TrustZone address space controller (*TZASC*): controls the partitioning of the dynamic random-access memory (DRAM) into distinct memory regions, and designates a memory region as secure or non-secure.
3. The TrustZone memory adapter (*TZMA*): provides similar functionalities to the *TZASC* for the on-chip static memory.
4. The direct memory access controller (*DMAC*): controls and prevents a peripheral assigned to the non-secure world from performing a DMA transfer to or from the secure world memory.
5. The generic interrupt controller (*GIC*): controls or interrupts partitioning into secure and non-secure worlds.
6. The L2 cache controller checks the *NS* bit corresponding to the cache request. This bit is then taken into account as the 33rd bit of the memory bus and indicates which world refers to the cache memory access.
7. AXI interconnect: if required, this block configures the 3rd party IP added by the SoC designer as secure or non-secure IP. A secure IP can be accessed by software running only in the secure world, while a non-secure IP can be accessed by software running in both the non-secure world and the secure world.

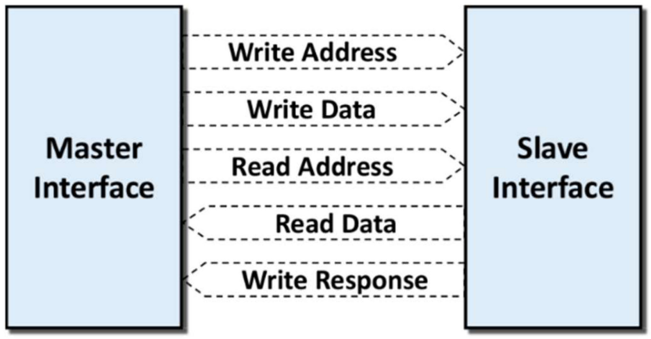

Fig. 1. The communication channels of the AMBA3 AXI system bus between a master interface and a slave interface.

The hardware blocks involved in the TrustZone technology ensure that the configured security separation is not violated during execution. Some of these hardware blocks have the capability of dynamic configuration, which allows the user to configure them in run time using memory registers. This last feature will be exploited in attack #6 presented in Section 6 in this paper.

## 4 Elements of AMBA3 AXI System Bus

To fully understand the attack scenarios presented in this paper, the reader needs details on the advanced extensible interface (AXI) bus that is part of the ARM advanced microcontroller bus architecture (AMBA) specifications.

Indeed, one of the most useful features of the ARM TrustZone technology is the ability to secure the full system and not only the processor core. This feature allows the system user to extend the security from the processor core to the full system including the processor memories and peripherals. In an ARM-based TrustZone-enabled system, the security extension is implemented using the AXI system bus.

### 4.1 The AXI Bus Communication Channels

The AXI system bus uses a handshake protocol between a master and a slave. As illustrated in Fig. 1, its interface consists of five channels.

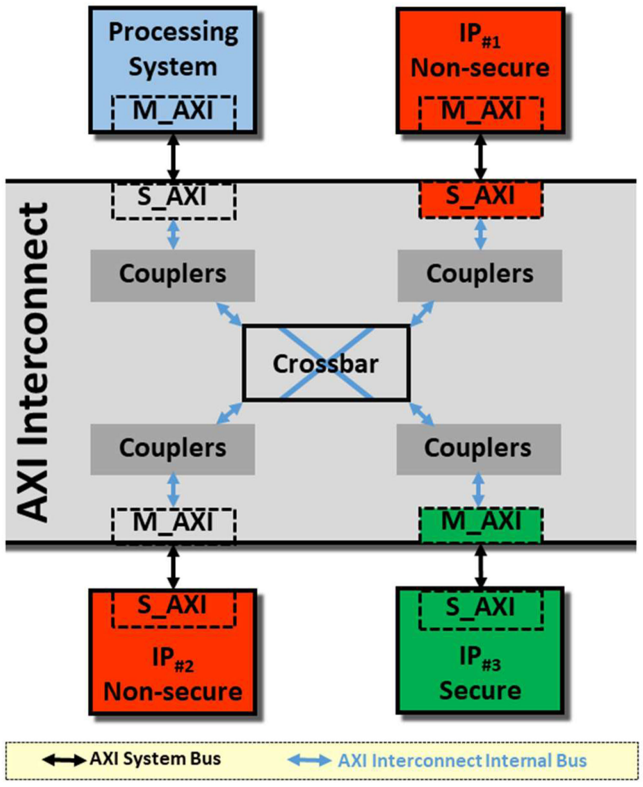

Fig. 2. Illustration of an AXI interconnect coupling one processing system (master executing secure or non-secure application), one master-IP (the non-secure IP$_{\#1}$) and two slave-IPs (the non-secure IP$_{\#2}$ and the secure IP$_{\#3}$).

Each of the five communication channels uses a *VALID* and *READY* handshake signal pair to indicate when the transmitter is ready to send valid data and when the receiver is ready to process bus data. For any transactions (address/data/response), the transmitter *VALID* and the receiver *READY* signals should be asserted (forced to a high level logic status).

As mentioned above, the read and write channels of the AXI system bus include both *NS* bits. The *ARPROT* bit is included in one of the two read channels and the *AWPROT* bit is included in one of the three write channels.

During a transaction (read or write), the communication master assigns an appropriate value to the dedicated *NS* bit. The slave checks the *NS* bit to be sure that no security violation has occurred during the transaction. For example, if a non-secure master attempts to obtain write access (the *NS* bit, *AWPROT*, the logic state is high) on a secure slave, the slave sends an error on the AXI bus system write response channel. To do so, it uses the two dedicated 2-bit signals *BRESP* (on the response channel) and *RRESP* (on the read data channel). This error could be a *DECERR* (decode error with the binary value "11") or a *SLVERR* (slave error with the binary value "10"). If the transaction is correct, the slave sends an *OK* on the AXI bus using the same 2-bit signals (with the binary value "00").

### 4.2 The AXI Interconnect

In a FPGA-based TrustZone-enabled heterogeneous SoC, an AXI interconnect is used to statically connect the hardware IPs configured in the FPGA fabric and the ARM-centric processing system. As illustrated in Fig. 2, on the AXI interconnect, each IP connected by an interface (slave or master interface as a function of the IP status) can be configured as a master or as a slave, and also as a secure IP or a non-secure IP. Inside the AXI interconnect, a crossbar routes the traffic between the slave and master interfaces of the AXI interconnect. Along each pathway connecting a slave interface or master interface to the crossbar, an optional series of AXI infrastructures (couplers) can perform various conversions and buffering functions.

During design time, the security feature on the AXI interconnect can be statically enabled or disabled and the AXI interconnect master interface slots can be assigned a static secure or non-secure status. Each AXI interconnect master interface slot is connected to a third-party hardware IP.

In the FPGA fabric of a TrustZone-enabled heterogeneous SoC, the crossbar of the AXI interconnect is usually responsible for checking the security status of the transaction by comparing the *NS* bits on the AXI system bus with the security status of the IP involved. In the case of a security violation, the crossbar sends back a *DECERR* error to the appropriate couplers and these then stop the transaction and transfer the error back to the master IP.

## 5 Attack Setup

This section first presents the threat models and second, the targeted SoC, a Xilinx Zynq-7010 device, and the targeted system prototype.

### 5.1 Threats Model

Designing and testing a heterogeneous hardware-software system on a complex SoC requires many engineers, and in most cases relies on third party IP and tools to meet market expectations. The more complex the system, the larger the threat model. Indeed, a rogue design engineer, a malicious third party IP or virus-infected computer aided design tools have the potential ability to alter the system security even when the TrustZone technology is used. There are many potential attack paths. Each malicious/infected component can maliciously change the system and affect its security.

The rest of this paper considers the threats that result from a malicious modification of the system design. Attacks exploiting such threats can lead to denial-of-service (DoS), to privilege escalation, to leakage of secure information, and to the installation of a malicious software.

### 5.2 Xilinx Zynq-7010 Programmable SoC Device

All the attack scenarios presented in this paper use the Xilinx Zynq-7010 SoC as target. This device is an TrustZone-enabled heterogeneous SoC widely used in most industry and academic use cases. Nevertheless, the Xilinx Zynq SoC closely resembles other heterogeneous SoCs, such as Intel Cyclone V SoC or Intel Stratix 10 SoC. The attack scenarios presented here are also conceivable with other SoC targets.

The Xilinx Zynq-7010 SoC is partitioned into an ARM-based *processing system*, and a FPGA fabric called *programmable logic*. The Xilinx Zynq-7010 *programmable logic* includes a hard-core cipher block used for authentication and decryption to ensure a secure configuration and a secure boot.

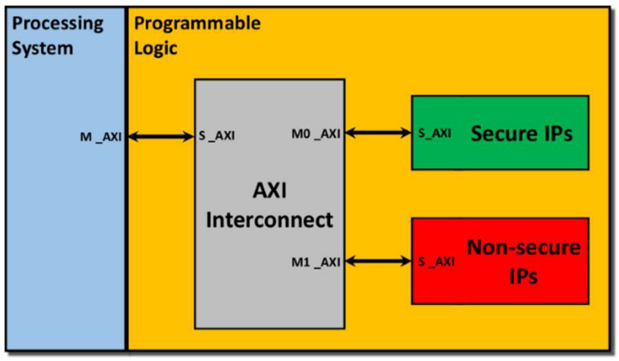

Fig. 3. Architecture of the simple system prototype used for the proof of concept of the attack scenarios.

The Xilinx Zynq-7010 is compliant with TrustZone technology but the software and hardware implementation of the TrustZone security services involves a complex process. Interested readers can follow the cost free on-line tutorial [40] on designing a TrustZone-enable system with the Xilinx Vivado CAD tool.

### 5.3 System Prototype

This paper presents the proof of concept of several attack scenarios. In order to perform the attacks, a prototype of a simple system was developed with the bare necessities. Fig. 3 presents the architecture of this simple system, which contains an ARM-centric *processing system* (with activation of the TrustZone features) and a *programmable logic*. The *programmable logic* embeds one AXI interconnect that links the *processing system* as a master (secure or not depending on the security status of the software application) with two hardware slave IPs. One IP is secure and the other is non-secure. The two hardware IPs have a dedicated memory region for configurations and data storage.

In the case of Xilinx SoC, the security check is done at the AXI interconnect level and is not transferred to the hardware IPs. The master of the system is responsible for checking the security of the transaction. The crossbar of the AXI interconnect does the security check just after receiving the address from the master, using a conditional test. The test compares the static security status of the targeted IP and the protection signal. The *NS* bits inform the security status of the all system items involved in a transaction.

## 6 Presentation of the Attacks

This section presents six attacks targeting the TrustZone propagation into the heterogeneous SoC.

### 6.1 Attack#1: To Compromise AXI *AWPROT/ARPROT* Communication Signals

The aim of this attack is to fix the logic state of the protection signals *AWPROT[1]/ARPROT[1]*. When the logic status of these signals is fixed at a low level, the connection between the AXI Interconnect and the *processing system* is still considered to be secure even when a software executed in the non-secure world wants to access a secure IP resulting in a privilege escalation attack. Indeed, the non-secure software accesses the secure IP as well as each memory region allocated to the secure IP with read and write permission.

When the logic status of the *AWPROT[1]/ARPROT[1]* is fixed at a high level, the connection between the AXI Interconnect and the *processing system* is always considered to be non-secure even when a software executed in the secure world wants to access a secure IP, which leads to a denial-of-service.

One way to perform this attack is to use malicious modifications of small hardware often presented as hardware Trojans in the literature [41], [42]. This Trojan payload is the AXI interconnect between one master interface and one slave interface. Fig. 5 shows three examples of malicious hardware modifications of the AXI interconnect that target the *AWPROT* signal (the same modifications can be used to compromise *ARPROT* signal). The hardware Trojans may or may not include a trigger.

Another solution is to automatically perform an RTL code modification after synthesis using a malicious TCL script generated by a compromised CAD tool. In the case of a CAD tool, the modification is compromised by using a dedicated malicious software (malware) design to target the CAD tool. However, only using a malicious software to attack the CAD tool when the SoC design is targeted is an efficient attack path. Moreover, the TCL script can also be altered by a malicious designer. The design rule check tool (DRC) and synthesis tools are both based on TCL script and thus constitute a major threat if the source of the TCL script is not trusted, as demonstrated in [43].

It is not difficult to change the TCL script. For example, to automatically achieve the hardware modification presented in Fig. 5a, the malicious entity (malicious software/rogue designer) has to add the two malicious TCL commands presented in Fig. 4. The first command disconnects the net bit#1 of the AXI interconnect AWPROT signal connected to the pin bit#1 of AWPROT of the AXI interconnect. The second command connects the disconnect pin to GND in order to fix it at low logic level. To hide the malicious TCL from the designer in the command log, the TCL script is executed using *–notrace* and the commands use the *-quiet* option to ignore command errors to avoid to leave traces (warning and errors in the case of failure to execute the malicious commands).

The limitation of this method of modifying TCL is that the system designer can change the name of the all nets before RTL code is generated. Nevertheless, Xilinx offers some commands that allow the CAD user (malicious or not) to obtain information about the design such as the net name that make it possible to get round this limitation by automatically adapting the TCL modifications.

```
disconnect_net -quiet -net [get_nets design/signal_AXI_AWPROT[1]] -objects [get_pins design/AXI_Interconnect/S_AXI_AWPROT[1]]
connect_net -hier -quiet -net design/<constant0> -objects [get_pins design/AXI_Interconnect/S_AXI_AWPROT[1]]
```

Fig. 4. Two TCL commands to add in the TCL script to automatically change the RTL code after synthesis and insert malicious hardware modifications.

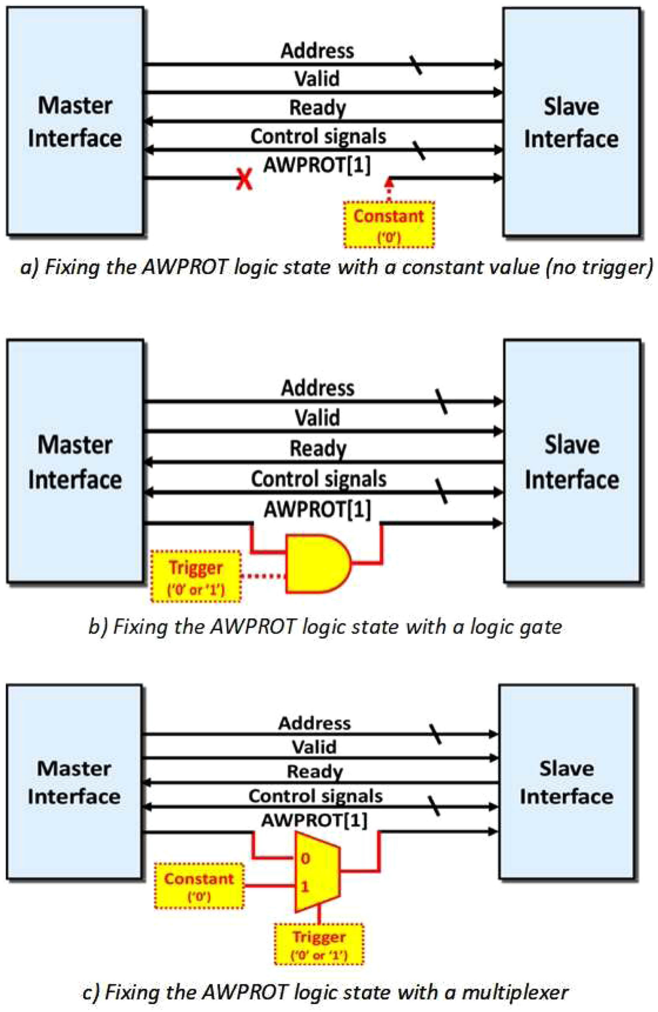
Master Interface
Address
Valid
Ready
Control signals
AWPROT[1]
Slave Interface
Constant ('0')
a) Fixing the AWPROT logic state with a constant value (no trigger)
Trigger ('0' or '1')
b) Fixing the AWPROT logic state with a logic gate
0
1
c) Fixing the AWPROT logic state with a multiplexer

Fig. 5. Three examples of modifications to the AXI interconnect targeting the *AWPROT* signal by malicious hardware.

## 6.2 Attack#2: To Compromise AXI *BRESP/RRESP* Response Signals

As presented at the end of Section 4.1, the 2-bit transaction response signals *BRESP* and *RRESP* are essential for the management of slave security. As a consequence, these signals are appropriate targets for attack. To this end, a dedicated multiplexer can be added on the *BRESP* or *RRESP* signal as illustrated in Fig. 6 (in the case of *BRESP* signal). The techniques of malicious hardware modification presented in the previous subsection are also valid in this context.

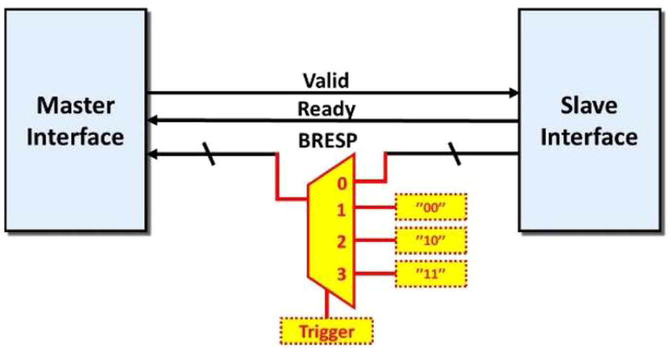

Fig. 6. An example modification of the BRESP signal by malicious hardware to force the transaction response from a slave interface.

Fig. 6 shows a malicious 4-to-1 multiplexer on the BRESP signal used to force the slave response, but depending on the attack scenario, this multiplexer could be smaller and capable of forcing only one or two possible responses.

When the BRESP signal is forced to the binary value "00" the slave response is forced to OK even in the case of security status violation (software executed in the non-secure world wants to access a secure IP), which leads to a privilege escalation. Indeed, the non-secure software accesses the secure IP and each memory region allocated to the secure IP with read and write permission.

When the BRESP signal is forced to the binary value "10" or "11", the slave response is forced to SLVERR or DECERR, respectively, even in the case of a correct transaction (when a software executed in the secure world wants to access a secure IP), which leads to a denial-of-service.

## 6.3 Attack#3: To Compromise the AXI Crossbar Verilog Code or LUT Configuration

The AXI communication 'signals are not the only sensitive items. As described in Section 4.2, the crossbar of the AXI interconnect plays an essential role in system security. Indeed, the crossbar is responsible for checking the security status of each transaction by comparing the *NS* bits on the AXI system bus (security status of the software executed by the *processing system)* with the security status of the targeted hardware IP in the *programmable logic*.

As described in Section 6.1, if a CAD tool is compromised (or in the case of a malicious designer), it is possible to attack the crossbar by changing its Verilog hardware description. Fig. 7 presents one possible malicious modification of this Verilog code. Only one Verilog line is changed by adding a conditional branch to obtain an escalade privilege when the transaction concerns a specific secure IP. In Fig. 7, the specific IP has the ID *"01"*, from now on referred to as *IP#01*. The aim of the modification is to force the *m_aerror_i[1]* to a low logical level, which indicates that the transaction is correct (a high logical level of this bit leads to an exception in the *processing system*).

In the modified Verilog code, the *target_mi_hot* signal is used to indicate the ID of the targeted IP. It is a vector where every bit is related to one of the master interfaces of the AXI Interconnect. For example in the case of two IPs (two master interface) connected to the AXI interconnect, the ID of the first master interface "01", the second is "10". The *P_M_SECURE_*

```
assign target_secure = |(target_mi_hot & P_M_SECURE_MASK);
…
assign any_error = C_RANGE_CHECK && (m_aerror_i != 0);
assign any_error_i[0] = ~match;
assign any_error_i[1] = target_secure && S_APROT[P_NONSECURE_BIT];
```

a) Original Verilog code of the AXI interconnect crossbar

```
assign target_secure = |(target_mi_hot & P_M_SECURE_MASK);
…
assign any_error = C_RANGE_CHECK && (m_aerror_i != 0);
assign any_error_i[0] = ~match;
assign any_error_i[1] = (target_mi_hot  == 2'b01) ? (1'b0)
                        : target_secure && S_APROT[P_NONSECURE_BIT];
```

b) Modified Verilog code of the AXI interconnect crossbar

Fig. 7. Malicious modification of the Verilog code of the AXI interconnect crossbar.

```
set_property INIT "32'h000000E0" [get_cells cell_name]
```

Fig. 8. Malicious TCL command to force the configuration of a LUT used for crossbar logical configuration.

*MASK* signal is also a vector of bits that indicate the security status of every master interface of the AXI Interconnect. If the bit is a high logical level the interface is secure.

The *target_secure* signal in the Verilog code is the result of the logical and bitwise between the *P_M_SECURE_MASK* and *target_mi_hot*, if the master interface is secure the result is equal to the interface ID, other the result is equal to "00".

The transactions to other IPs than the *IP#01* are the same as with the original Verilog code. Indeed, the *m_aerror_i[1]* bit depends on the result of the logical and between the *NS* bit value (described by the *S_APROT_NONSECURE_BIT* signal in the Verilog code) and the IP security status (described by the value of the *target_secure* signal in the Verilog code). Unlike the *NS* bit, the *S_APROT_NONSECURE_BIT* state is at a high logical level when the IP is secure and at a low logical level when the IP is non-secure).

When the transaction targets the *IP#01*, the *m_aerror_i[1]* bit is forced to a low logical level. The connection between the AXI Interconnect and the *processing system* is still considered to be secure even when a software executed in the non-secure world wants to access the secure *IP#01*. Consequently, this attack results in a privilege escalation. This kind of modification could be used to create a denial-of-service if the *m_aerror_i[1]* is forced to high logical level.

Another possible form of attack is changing the FPGA configuration at the LUT level. Indeed, the data stored in the LUT are responsible for the configuration of the LUT and hence for the logical function between the inputs and outputs of the LUT. By observing the modifications in the configuration between several instances of the AXI interconnect block, the attacker can find information on the LUTs linked to the crossbar. Despite the configuration of the AXI interconnect, these LUTs conserve the same logical configuration, so the attack can target them. Indeed, the configuration of the LUTs can be forced to a null logical function which gives a low logic level to its output despite the state of the input. To do so, the attacker uses the malicious TCL command presented in Fig. 8. With this configuration, the crossbar of the AXI interconnect block will respond *OK* despite the security status of the hardware IP involved in the communication. As in the previous example, this attack leads to a privilege escalation.

## 6.4 Attack#4: To Change the Static Security Status of an IP from the AXI Interconnect Block

After the AXI communication signal and the crossbar, it is possible to directly change the security status of an IP automatically in the AXI interconnect by using a malicious TCL command (in the case of a compromised CAD tool or of a malicious designer). To do so, the attacker hides the malicious TCL command presented in Fig. 9 in the synthesis TCL script.

```
set_property CONFIG.M00_Secure true [get_bd_cells AXI_Interconnect]
```

Fig. 9. Malicious TCL command to hide in the synthesis TCL script by the attacker in order to force the security-status of the AXI interface *M00* to non-secure even if the linked IP is secure.

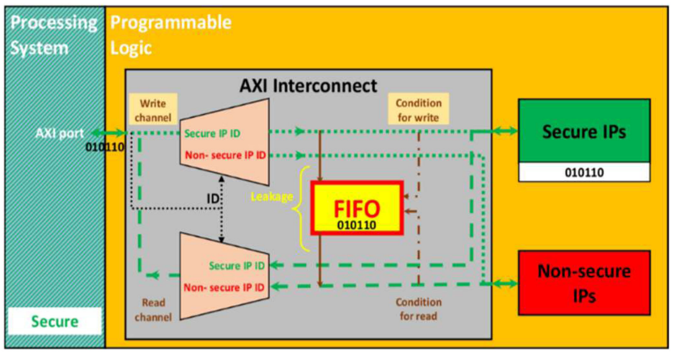

Fig. 10. Malicious insertion of a FIFO in the AXI interconnect block to spy on the data sent from the *processing system* (in the secure world) to the secure IP.

This command changes the static security status of a master interface slot of the AXI interconnect (linked to a slave interface of a non-secure IP) to non-secure. The Xilinx AXI interconnect block offers 16 master interface slots to hardware IPs connections. The attacker can target all 16 slots and change their security status to non-secure. With such a change in the security status, the IP is still considered by the AXI interconnect block to be non-secure, thereby allowing all software to access it even if the software is executed in the non-secure world. This attack also leads to a privilege escalation.

Note that to perform this privilege escalation attack, the attacker needs to know the exact names of the cells in the design in order to target the correct AXI Interconnect. To do so, Xilinx proposes many TCL commands to automatically obtain this information. The attacker can then switch the static security status of all the master AXI Interconnect interface slots.

## 6.5 Attack#5: To Insert Malicious FIFO in the AXI Interconnect Block

In 2016, Fern et al. first presented insertion of a malicious FIFO in an AXI interconnection system [44]. Their paper described the insertion of a hardware Trojan in an AXI4-Lite in detail but assumed that the method was adaptable to any bus. Based on this first study, we propose a malicious modification of the RTL code of the AXI Interconnect Xilinx block in order to add a FIFO inside the communication path. The FIFO spy on the data sent from the software executed in the secure world by the *processing system* to the secure IP and, if requested, provides these data to the non-secure IP. However, it is also possible to use complementary spy circuitries such as the lightweight electromagnetic transmitter proposed by Bossuet et al. [36] to discretely send the data outside the SoC. The quantity of leaked data depends in particular on the depth of the FIFO, but the FIFO needs to be as small as possible to ensure the success of this attack.

The attacker makes the malicious modification to the AXI Interconnect as illustrated in Fig. 10. Fern et al. [44] already presented the details on how the FIFO is integrated in the AXI Interconnect and explained how to choose read and write conditions for the FIFO. In our case, we used the *valid signal* of the write data channel connected to the secure IP block as a FIFO write condition. We also used the *valid signal* of the write data channel connected to the non-secure IP block as a FIFO read condition.

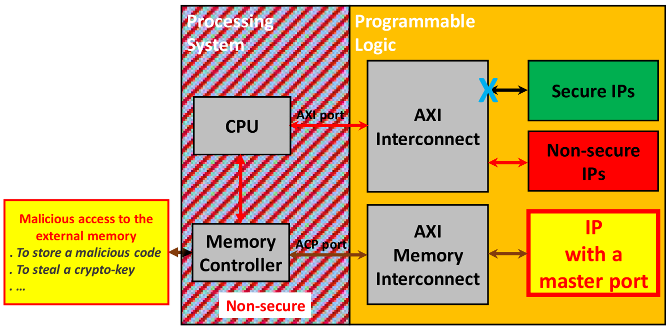

Fig. 11. System architecture embedding a malicious IP (in yellow) with direct access to the external memory exploiting the ACP.

The attacker can add a condition to stop storing data in the FIFO, for example, the *WLAST* signal that indicates the end of a write transaction in the AXI protocol can be used.

### 6.6 Attack#6: To Directly Access the External Memory

This efficient attack uses a malicious hardware IP with direct access to the external memory using the ARM accelerator coherency port (ACP) available on Xilinx Zynq [45] but also in other SoC such as Intel SoC-FPGA [46]. The ACP allows bus mapping between the malicious hardware IP and the whole memory range including the ARM TrustZone configuration registers such as the configuration registers of the *TZPC* and the *TZASC* (see Section 3). In this way, the malicious IP bypasses almost all the security measures. It can read secure data, steal a cipher key, or install a malware.

Fig. 11 illustrates the uses of system architecture for this attack. In the figure, the *processing system* runs a software in the non-secure world, consequently, the *processing system*, the secure and non-secure IPs do not have access to the external memory. Only the malicious IP has full direct access to the external memory including the configuration register space. The malicious IP is an AXI master; it controls the communication channel linking them to the *processing system*. As an AXI master, the malicious IP also controls the security status of the AXI bus; the processing system security status (secure or non-secure world) does not affect it.

The success of this attack depends on two elements. The first is how the malicious IP is mapped in the memory? It needs to have access to the targeted region of the external memory (storage of secure data, storage of instructions to store the malware, storage of configuration registers). The second element depends on the attacker's knowledge of software mapping and of the memory mapping of the targeted platform.

For example, to target the ARM *TZPC* and *TZASC* in the Zynq-7010 SoC, the attacker must know that the address of the *TZPC* configuration register is 0xE0200018, and that the address of the *TZASC* configuration register is 0x F8000430. Each bit of the *TZPC* configuration register controls the security status of one peripheral in the Zynq-7010 SoC. Similarly, each bit of the *TZASC* configuration register controls the security status of a 64 MB region of the memory. For both TrustZone controllers, the bit is set to a high logical level when the peripheral or the memory region is non-secure, other ways are secure. In normal circumstances, the TrustZone controllers are configured during the boot process and at run time only for specific application requests from the secure world. But the malicious IP is able to change the TrustZone controllers at run time at any time.

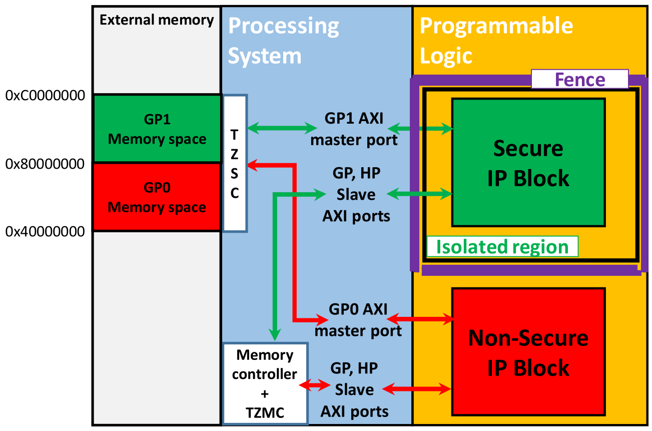

Fig. 12. Architectural results of the proposed methodology and the two new controllers. Two isolated IP blocks (secure and non-secure) in the *programmable logic* are connected to the memory through distinct AXI ports and the two new controllers: TZSC and TZMC.

## 7 Avoiding Attacks

In the previous section, we described efficient attack paths that exploit a hardware Trojan, a malicious third party IP, a corrupted CAD or a malicious designer. In this section, we first present and discuss a design methodology to increase the security of FPGA-based heterogeneous SoC. The methodology has never been previously presented. The four steps of the method should be followed exactly to perfectly isolate a secure zone and a non-secure zone in the *programmable logic*. Next, we propose to add two new controllers to the *processing system* of the SoC and to use a design rule checker. The methodology, the proposed new controllers and the design rule checker provide high-level and holistic protection against the attacks described above. Fig. 12 provides an overview of the architectural results of the proposed methodology and the two new controllers described in the following sections.

### 7.1 A Design Method to Isolate the *programmable logic*

To avoid the attacks described in this paper, the designer needs to isolate the secure and non-secure IPs in the *programmable logic*. The security isolation requires distinct *processing* system AXI ports, as shown in Fig. 12. The design can be used for both Intel Cyclone V and Xilinx Zynq, but a specific methodology must be respected.

First, in the specific case of Xilinx Zynq SoC, to isolate the secure and non-secure IPs in the *programmable logic*, the designer should use Xilinx's isolation design flow (IDF) methodology, which allows independent IPs to operate on a single FPGA with logical and physical separation. The IDF methodology places and routes IPs in an isolated zone of the FPGA surrounded by a fence (a set of unused FPGA elements in which no routing or logic is present). The IDF also uses an independent design rule check tool to check the fence and the connection to or from the isolated zone.

With the IDF methodology, the designer creates two separate IP blocks, a secure one (called the *secure IP block*) and a non-secure one (called the *non-secure IP block*). The two IP blocks have separate resources (output/input pins, clocks, etc.) and the secure IP block is placed in an isolated zone, as close as possible to the *processing system* AXI ports to avoid any tampering with the bus security status (thereby avoiding hardware Trojan and malicious IPs).

Second, the two separate IP blocks have to be connected to the memory using the *processing system* AXI slave and master ports. The memory stores the configuration of each IP interface (*secure / non-secure*). The Xilinx Zynq-7010 SoC has two general-purpose AXI master ports (labelled *GP0 AXI port* and *GP1 AXI port* in Fig. 12). Each has a dedicated memory space (labelled *GP0 Memory space* and *GP1 memory space* in Fig. 12). If the AXI port configuration is *secure*, the memory space related to the port is not accessible by a software application executed in the non-secure world.

Third, the *processing system* of the Xilinx Zynq-7010 SoC also has two GP AXI slave ports (labelled *GP0 slave AXI port* and *GP1 slave AXI port* in Fig. 12), four high performance (HP) ports, and one ACP (previously used in attack#6). All these ports allow direct access to the entire memory or, alternatively, only to specific regions. The *processing system* does not directly control access to the memory ports. Consequently, the designer has to configure the GP slave ports and the HP ports to only access the non-secure memory or to access all the memories, from the *programmable logic*. The configuration registers related to the slave interfaces of the secure IPs are mapped in the GP1 memory space (with the addresses from 0x80000000 to 0xC0000000). The designer has to configure the GP1 AXI master port as *secure* to stop the propagation of non-secure requests. This prevents all access from software executed in the non-secure world to the GP1 memory space. The design has to configure the GP0 AXI master port as *non-secure* to allow access to GP0 memory space from software executed in the non-secure world or executed in the secure world.

Fourth, the designer has to configure the slave AXI ports (the HP ports and the GP slave ports) connecting the non-secure IPs to the memory as *non-secure* to restrict access to only the non-secure memory.

These four-step methodologies allow designers to reinforce their design against the attacks presented above. However, this is still not sufficient, and additional protections should be used to protect the system against malicious hardware. In the following, two new processing system controllers and a design rule checker are presented as additional protection.

## 7.2 Two New TrustZone Controllers

In this section, we introduce two new TrustZone controllers of the *processing system*: the TrustZone slave controller (TZSC) and the TrustZone master controller (TZMC). The software configuration of the TZSC and the TZMC can only be done in the secure world.

The TZSC functions as a gatekeeper for the memory space allocated to the slave interfaces of the *programmable logic*. The TZSC checks the security status of each memory access from a slave interface and generates exceptions when security rules are violated. The TZSC is a mitigation of all the previously presented attacks except attack#6. The TZSC is placed between the memory spaces and the AXI ports, as shown in Fig. 12. The TZSC addresses two memory zones; a static memory zone that is configured only once during secure boot, and a dynamic memory zone with the capability of configuring run-time software. This run-time software configuration is executed only in the secure world. The dynamic memory zone allows a software application executed in the secure world to use a non-secure IP for a limited time and for a specific function only.

The TZMC protects and partitions a new memory space dedicated to direct memory access. In this way, direct memory access with no security checking is prohibited. Attack#6 (Section 6.6) revealed the weakness of a TrustZone-enabled system against direct malicious memory access from the programmable logic. The TZMC is a mitigation of this attack. The TZSC is placed between the memory space dedicated to direct memory access and the AXI ports, as in Fig. 12. In the specific case of Xilinx Zynq SoC, the design of the TZMC block can be based on the Xilinx Memory Protection Unit [47].

TZSC and TZMC are embedded in the processing system, so hardware costs for the SoC are not significant. Moreover, as the two controllers release the AXI interconnect from security checking, the performance of the system is not reduced.

## 7.3 Checking Post-Synthesis Security Design Rules

Most of the attacks presented in Section 6 of this paper could be performed by exploiting a CAD tool corrupted by a malicious software (using dedicated viruses) or directly by a malicious designer. To protect the system against such threats, the CAD tool requires dedicated software protection, and the design should be protected by checking post-synthesis security design rules. For the last point, solutions are available in the recent literature. However, most of the proposed solutions are limited to hardware Trojan avoidance, [3], [48], [49] and the security of third-party hardware IPs [50], [51].

The CAD tool has to check the security design rules of the connections between the AXI interfaces. It has to check that not every signal in the AXI channels is corrupted. To do so, in the case of the Xilinx Vivado CAD tool, we propose a python-programmed design rule checker (the source code is freely available online [52]). Despite the fact it is a dedicated code for Xilinx Vivado CAD tool, this *python* code can easily be adapted to other CAD tools. The security checker takes the VHDL code source of the top level design with the AXI interconnect component instantiation as an input. As output, the security checker reports the corrupted connections (if necessary) together with the name of the affected signal and interface. The security checker begins by parsing the VHDL source code and sorting the instantiated components and internal signals. For each component, the security checker lists its interfaces with their connections from/to signals. The security checker then takes the connected components two-by-two and checks the signals between them. This solution should be systematically added to the CAD tool to insure the integrity of the design RTL source code and to avoid malicious modification of the IP HDL code source code.

## 8 Conclusion

The security of FPGA-based heterogeneous SoC is mandatory to enable embedded system designers to develop secure execution environments for sensitive applications. The extension of the ARM TrustZone technology in the SoC appears to be a good solution if security is guaranteed. However, the extension of the ARM TrustZone is currently not secure. The present work provides proof that many threats exist and that it is possible to use them to attack system security, possibly with dramatic consequences. This paper describes six attacks that were successfully carried out on a typical software-hardware system, but other attack paths are possible. Consequently, the designer who wishes to develop sensitive applications on such heterogeneous SoC must take the security of the entire system into account as early as possible in the design flow. This paper presents some new security solutions and design recommendations for designers. Unfortunately, most designers are not accustomed to security problems, nevertheless, CAD tools should offer integrated and automatic solutions such as those presented in this paper.

## References

[1] T. Alves and D. Felton, “TrustZone: Integrated Hardware and Software Security,” ARM White Paper, Jul. 3, 2004.

[2] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, “Providing root of trust for arm trustzone using on-chip sram,” in *Proc. 4th Int. Workshop Trustworthy Embedded Devices*, 2014, pp. 25–36.

[3] V. Jyothi, M. Thoonoli, R. Stern, and R. Karri, “FPGA TrustZone: Incorporating trust and reliability into FPGA designs,” in *Proc. IEEE 34th Int. Conf. Comput. Des.*, 2016, pp. 600–605.

[4] K. Eguro and R. Venkatesan, “FPGAs for trusted cloud computing,” in *Proc. 22nd Int. Conf. Field Programmable Logic Appl.*, 2012, pp. 63–70.

[5] M. A. Will and R. K. L. Ko, “Secure FPGA as a service-towards secure data processing by physicalizing the cloud,” in *Proc. IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 449–455.

[6] A. Salman, M. Rogawski, and J. Kaps, “Efficient hardware accelerator for IPSec based on partial reconfiguration on Xilinx FPGAs,” in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2011, pp. 242–248.

[7] B. Driessen, T. Güneysu, E. B. Kavun, O. Mischke, C. Paar, and T. Pöppelmann, “IPSecco: A lightweight and reconfigurable IPSec core,” in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2012, pp. 1–7.

[8] M. Grand, L. Bossuet, G. Gogniat, B. Le Gal, J. Delahaye, and D. Dallet, “A reconfigurable multi-core cryptoprocessor for multichannel communication systems,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops PhD Forum*, 2011, pp. 204–211.

[9] M. Barbareschi, E. Battista, V. Casola and A. M. E. N. Mazzocca, “On the adoption of fpga for protecting cyber physical infrastructures,” in *Proc. 8th Int. Conf. P2P Parallel Grid Cloud Internet Comput.*, 2013, pp. 430–435.

[10] V. Kharchenko, A. Kovalenko, O. Siora, and V. Sklyar, “Security assessment of FPGA-based safety-critical systems: US NRC requirements context,” in *Proc. Int. Conf. Inf. Digital Technol.*, 2015, pp. 132–138.

[11] A. Singh, A. Prasad, and Y. Talwar, “SCADA security issues and FPGA implementation of AES - A review,” in *Proc. 2nd Int. Conf. Next Generation Comput. Technol.*, 2016, pp. 899–904.

[12] B. El Mehdi, C. Marchand, L. Bossuet, and A. Aubert, “On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC,” in *Proc. 30th IEEE Int. Syst. Chip Conf.*, 2017, pp. 108–113.

[13] D. Rosenberg, “Qsee trustzone kernel integer over flow vulnerability,” in *Proc. Black Hat Conf.*, 2014, p. 26.

[14] Y. Chen, Y. Zhang, Z. Wang, and T. Wei, “Downgrade attack on TrustZone,” arXiv:1707.05082, 2017.

[15] F. Majéric, B. Gonzalvo, and L. Bossuet, “JTAG fault injection attack,” *IEEE Embedded Syst. Lett.*, vol. 10, no. 3, pp. 65–68, Nov. 2017.

[16] M. Lipp, “Cache attacks and rowhammer on arm,” Master Thesis, Graz University of Technology: Graz, Austria, Oct. 2016.

[17] P. Carru, Attack trustzone with rowhammer, GreHack, 2017, https://grehack.fr/data/2017/slides/GreHack17_Attack_TrustZone_with_Rowhammer.pdf.

[18] A. Tang, S. Sethumadhavan, and S. Stolfo, “CLKSCREW: Exposing the perils of security-oblivious energy management,” in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1057–1074.

[19] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani, “DFS covert channels on multi-core platforms,” in *Proc. IEEE Int. Conf. Very Large Scale Integr.*, 2017, pp. 1–6.

[20] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, “Thermal covert channels on multi-core platforms,” in *Proc. 24th USENIX Security Symp.*, 2015, pp. 865–880.

[21] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, “TruSpy: Cache side-channel information leakage from the secure world on ARM devices,” *IACR Cryptology ePrint Archive*, vol. 2016, 2016, Art. no. 980.

[22] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, “ARMageddon: Cache attacks on mobile devices,” in *Proc. 25th USENIX Security Symp.*, 2016, pp. 549–564.

[23] X. Zhang, Y. Xiao, and Y. Zhang, “Return-oriented flush-reload side channels on arm and their implications for android devices,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 858–870.

[24] M. Green, L. Rodrigues-Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, “AutoLock: Why cache attacks on ARM are harder than you think,” in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1075–1091.

[25] S. K. Bukasa, R. Lashermes, H. Le Bouder, J.-L. Lanet, and A. Legay, “How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip,” in *Proc. IFIP Int. Conf. Inf. Security Theory Practice*, 2017, pp. 93–109.

[26] J. Longo, E. De Mulder, D. Page and M. Tunstall, “SoC it to EM: Electromagnetic side-channel attacks on a complex system-on-chip,” in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2015, pp. 620–640.

[27] F. Majéric, E. Bourbao, and L. Bossuet, “Electromagnetic security tests for SoC,” in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, 2016, pp. 265–268.

[28] L. Riviere, Z. Najm, P. Rauzy, J.-L. Danger, J. Bringer, and L. Sauvage, “High precision fault injections on the instruction cache of ARMv7-M architectures,” in *IEEE Int. Symp. Hardware Oriented Secur. and Trust (HOST)*, May 2015, pp. 62–67.

[29] N. Timmers, A. Spruyt and M. Witteman, “Controlling PC on ARM using fault injection,” in *Proc. Workshop Fault Diagnosis Tolerance Cryptography*, 2016, pp. 25–35.

[30] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, “Laser-induced fault injection on smartphone bypassing the secure boot,” in *Proc. Workshop Fault Diagnosis Tolerance Cryptography*, 2017, pp. 41–48.

[31] B. Badrignans, J. L. Danger, V. Fischer, G. Gogniat and L. Torres, *Security Trends for FPGAS: From Secured to Secure Reconfigurable Systems*, Berlin, Germany: Springer, 2011.

[32] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*, Berlin, Germany: Springer, 2011.

[33] L. Bossuet, V. Fischer, L. Gaspar, L. Torres, and G. Gogniat, “Disposable configuration of remotely reconfigurable systems,” *Microprocessors Microsyst.*, vol. 39, pp. 382–392, 2015.

[34] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, “An inside job: Remote power analysis attacks on FPGAs,” in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2018, pp. 1111–1116.

[35] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, “Leaky wires: Information leakage and covert communication between FPGA long wires,” in *Proc. Asia Conf. Comput. Commun. Security*, 2018, pp. 15–27.

[36] L. Bossuet, V. Fischer, and P. Bayon, “Contactless transmission of intellectual property data to protect FPGA designs,” in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr.*, 2015, pp. 19–24.

[37] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, and G. Sigl, “How to break secure boot on FPGA SoCs through malicious hardware,” in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer 2017.

[38] J. Winter, “Experimenting with ARM TrustZone–Or: How i met friendly piece of trusted hardware,” in *Proc. IEEE 11th Int. Conf. Trust Security Privacy Comput. Commun.*, 2012, pp. 1161–1166.

[39] R. Rijswijk-Deij and E. Poll, “Using trusted execution environments in two-factor authentication: Comparing approaches,” *Open Identity Summit*, vol. 223, pp. 20–31, 2013.

[40] L. B. E. M. Benhani, “Design a TrustZone-enalble SoC usign Xilinx VIVADO CAD tool,” Technical Report, University of Lyon, Lyon France, 2017. [Online]. Available: http://labh-curien.univ-st-etienne.fr/~bossuet/VIVADO_TrustZone_tutorial.pdf, 2017.

[41] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, “Trustworthy hardware: Identifying and classifying hardware trojans,” *Comput.*, vol. 43, pp. 39–46, 2010.

[42] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, “Hardware Trojan attacks: threat analysis and countermeasures,” *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[43] Z. Zhang, Q. Yu, L. Njilla, and C. Kamhoua, “FPGA-oriented moving target defense against security threats from malicious FPGA tools,” in *Proc. IEEE Int. Symp. Hardware Oriented Security Trust*, 2018, pp. 163–166.

[44] N. Fern, I. San, C. K. Koc, and K.-T. Cheng, “Hardware trojans in incompletely specified on-chip bus systems,” in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2016, pp. 527–530.

[45] M. Sadri, C. Weis, N. Wehn, and L. Benini, “Energy and performance exploration of accelerator coherency port using Xilinx ZYNQ,” in *Proc. 10th FPGAworld Conf.*, 2013, Art. no. 5.

[46] Altera Corporation, “Hardware acceleration in SoC FPGAs,” Architecture Brief, Altera Technical Report, pp. 1–3, 2014.

[47] Xilinx, “Zynq ultraScale+ MPSoC Software devloper guide,” in Xilinx Technical Report, UG1137, pp. 119–120, 2018.

[48] X. T. Ngo, J.-L. Danger, S. Guilley, Z. Najm, and O. Emery, “Hardware property checker for run-time hardware trojan detection,” in *Proc. Eur. Conf. Circuit Theory Des.*, 2015, pp. 1–4.

[49] K. Xiao, A. Nahiyan, and M. Tehranipoor, “Security rule checking in IC design,” *Comput.*, vol. 49, pp. 54–61, 2016.

[50] E. Love, Y. Jin, and Y. Makris, “Enhancing security via provably trustworthy hardware intellectual property,” in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust*, 2011, pp. 12–17.

[51] B. Sherman, M. Borza, B. Rosenberg, and C. Qi, “Security assurance guidance for Third-Party IP,” *J. Hardware Syst. Security*, vol. 1, pp. 38–55, 2017.

[52] [Online]. Available: http://labh-curien.univ-st-etienne.fr/~bossuet/Security_design-rules_checker.zip

**M. Benhani** received the MSc degree in electronics and embedded systems from the Bordeaux Institute of Technology, Bordeaux, France, in 2016. He is currently working toward the PhD degree in Hubert Curien Laboratory, the University of Lyon. His research interests include hardware and embedded system security.

**L. Bossuet** received the MSc degree in electrical engineering from INSA, Rennes, France, in 2001, and the PhD degree in electrical engineering and computer sciences from the University of South Britanny, Lorient, France, in 2004. From 2005 to 2010, he was associate professor, and head of the Embedded System Department at Bordeaux Institute of Technology. From 2010 to 2017, he was associate professor at the University of Lyon/Saint-Etienne and currently holds the special CNRS (*Centre National de la Recherche Scientifique*) chair of Applied Cryptography and Embedded System Security. Since 2017, he has been professor at the University of Lyon/Saint-Etienne, where he is head of the computer science department of the Hubert Curien Laboratory. He is also head of the secured embedded systems and hardware architecture group of this laboratory. In 2016, he received the General Ferrié Award in electronics from the French SEE for his contribution to protection against IC counterfeiting and the IP protection. His main research focus is the security of embedded systems, IP protection, PUF design and characterization, secure-by-design crypto-processor, and reconfigurable architecture. He has published more than 150 refereed publications in these areas and is a senior member of the IEEE.

**A. Aubert** received the MSc degree in electrical engineering from INSA, Lyon, France, in 1998, and the PhD degree in electrical engineering from the INSA, Lyon, France, in 2001. In 2004, he became associate professor at the University of Lyon/Saint-Etienne. His main research focus is the security of embedded systems, in particular in design and characterization of True Random Numbers Generator (TRNG) implemented in FPGA and ASIC.