Security FPGA Analysis

Eduardo Wanderley, Romain Vaslin, Jérémie Crenne, Pascal Cotret, Jean-Philippe Diguet, Jean-Luc Danger, Philippe Maurine, Viktor Fischer, Benoit Badrignans, Lyonel Barthe, et al.

Chapter 2
Security FPGA Analysis

E. Wanderley, R. Vaslin, J. Crenne, P. Cotret, G. Gogniat, J.-P. Diguet, J.-L. Danger, P. Maurine, V. Fischer, B. Badrignans, L. Barthe, P. Benoit, and L. Torres

Abstract Security is becoming since several years a major issue in the domain of embedded systems. Fine grain reconfigurable architectures like FPGAs are providing many interesting features to be selected as an efficient target for embedded systems when security is an important concern. In this chapter we propose an overview of some existing attacks, a classification of attackers and the different levels of security as promoted by the FIPS 140-2 standard. We identify the main vulnerabilities of FPGAs to tackle the security requirements based on the security pyramid concept. We propose a presentation of some existing countermeasures at the different levels of the security pyramid to guarantee a defense-in-depth approach.

2.1 Introduction

Standardized cryptographic algorithms, like AES or RSA, are designed to resist cryptanalysis attacks, such as differential cryptanalysis. Only exhaustive attacks against the cipher key are possible, so the security of these algorithms relies on the length of the cipher key and if it is sufficiently long, such attacks are then impossible. Table 2.1 summarizes current minimum strength recommendations for cryptographic algorithms. In 2011, a minimum of 112 security bits is required for all cryptographic algorithms [5].

Thus cryptography algorithms are mathematically designed to be strong enough for current processing technologies. However, hackers can also attack software or hardware implementations for a lower cost. In such cases, implementation may be the weak point of the encryption process. For example, a new software side channel attack, called Branch Prediction Analysis (BPA) attack, was recently discovered and shown to be practically feasible on popular commodity PC platforms. This shows that a carefully written spy process, run simultaneously with an RSA process, can collect almost all the secret key bits in a *single* RSA signing execution. For this reason, security needs to be considered at different levels, i.e. from the technology to the application. If all these levels and the links between them are not taken into

G. Gogniat (✉)
Lab-STICC—UMR CNRS 3192, Bretagne-Sud University, Lorient, France
e-mail: guy.gogniat@univ-ubs.fr

Table 2.1 NIST recommendation for cryptographic algorithms

Date	Min of strength	Symmetric key algorithms	Asymmetric (RSA)	Hash (A)	Hash (B)
2006 to 2010	80	2TDEA*	1024	SHA-1**	SHA-1
		3TDEA*		SHA-224	SHA-224
		AES-128		SHA-256	SHA-256
		AES-192		SHA-384	SHA-384
		AES-256		SHA-512	SHA-512
2011 to 2030	112	3TDEA*	2048	SHA-224	SHA-1
		AES-128		SHA-256	SHA-224
		AES-192		SHA-384	SHA-356
		AES-256		SHA-512	SHA-384
					SHA-512
>2030	128	AES-128	3072	SHA-256	SHA-1
		AES-192		SHA-384	SHA-224
		AES-256		SHA-512	SHA-356
					SHA-384
					SHA-512

Hash (A): Digital signatures and hash-only applications

Hash (B): HMAC, Key Derivation Functions and Random Number Generation. The security strength for key derivation assumes that the shared secret contains sufficient entropy to support the desired security strength. The same remark applies to the security strength for random number generation

*TDEA (Triple Data Encryption Algorithm). The assessment of at least 80 bits of security for 2TDEA is based on the assumption that an attacker has 240 matched plaintext and ciphertext blocks at the most

**SHA-1 has been shown to provide less than 80 bits of security for digital signatures; the security strength against collisions is assessed at 69 bits. The use of SHA-1 is not recommended for the generation of digital signatures in new systems; new systems should use one of the larger hash functions. SHA-1 is included here to reflect its widespread use in existing systems, for which the reduced security strength may not be of great concern when only 80 bits of security are required

consideration, weaknesses can easily and rapidly appear in the devices concerned. Classical implementations of cryptography algorithms and secure embedded systems have been performed on ASICs and processors, but FPGAs are becoming increasingly attractive for cost and performance reasons and should be considered as a new alternative for security issues. As we explain in this chapter, FPGAs provide several key features that are recommended for security. For example when SRAM technologies are used, it is possible to dynamically change the functionality of the system to react to an attack. It is also possible to update some new hardware cryptography cores by remote reconfiguration even when the system has already been used for several years. This helps maintain the security level of a system at a time when cryptanalysis techniques are undergoing constant improvement. FPGAs pro-

vide an appropriate performance level for most embedded systems and, when combined with a processor core, can provide a whole secure solution. In this chapter we present an extensive analysis of FPGAs and their security in order to define the role that this technology could play in future applications.

The rest of the chapter is organized as follows: in Sect. 2.2, to give the reader some general background in the field, we describe the security principles and perform an initial analysis of attacks against FPGAs. In Sect. 2.3, we describe the security requirements for cryptographic modules according to the Federal Information Processing Standard Publication (FIPS PUB 1402) [33]. This point is very important when building a secure system. In Sect. 2.4, we analyze the vulnerabilities of FPGAs according to the security pyramid and describe possible technology, logic, architecture and system levels. In Sect. 2.5, we describe several countermeasures that have been developed and demonstrate how they can be used to build a secure system. Finally in Sect. 2.6, we present a number of conclusions.

2.2 Security Principles and Attacks Against FPGAs

The five main principles on which security is based to ensure the correct execution of a program and the correct management of the communications are:

- *Confidentiality*: only the entities involved in the execution or the communication have access to the data;
- *Integrity*: the message must not be damaged during transfer and the program must not be altered before being executed;
- *Availability*: the message and/or the program must be available;
- *Authenticity*: the entity must be sure that the message comes from the right entity and/or the system must trust the program source code;
- *Non-repudiation*: the entities involved in the execution or the communication must not be able to deny the exchange.

Guaranteeing all these points in a system is intellectually and financially costly. Efforts by attackers to disrupt one of these elements use two different approaches: extracting secret information (or the keys used to codify the information); or disturbing the system. The latter can also be classified in several levels: stop the system; temporarily stop the system; and/or change the functionality of the system (sometimes by opening doors to retrieve secret information).

Attackers are considered as adversaries, with varying abilities and with varying financial means at their disposal, and their attempts to disturb a system must be stopped. Generally, the power of an attacker can be classified using the description provided by IBM [1], as described into Chap. 1.

2.2.1 Hardware Attacks

The main goal of hardware attacks depends on the goal of the attacker. There are generally several objectives. The first is obtaining secret information like cipher keys. The second is causing a breakdown of the system (e.g. denial of service attack). We first describe attacks that aim to obtain secrets, and second, denial of service attacks. Some attacks are difficult to classify, hardware modification of the main memory being one of them since this kind of attack can be considered as a software attack but relies on a hardware technique to modify the memory content. The goal of this attack is to insert a malicious program in the system. A similar attack targets FPGAs by altering the bitstream.

To be able to decrypt information, the attacker needs the cipher key. One way to obtain cipher keys is to listen to side channels. This kind of attack is called a side channel attack and can take several different forms [19]. The best known relies on the power signature of the algorithm [25]. By analyzing the algorithm signature it is possible to infer the round of the algorithm. What is more, differential analysis combined with a statistical study of the power signature can lead to the extraction of the cipher key. However to reach this goal, the attacker has to make certain assumptions about the value of the key. The two methods are called SPA: Simple Power Analysis and DPA: Differential Power Analysis. Similar solutions are also possible using electromagnetic emissions (Differential Electromagnetic Analysis) [3]. Instead of analyzing the power signature, the attacker analyzes electromagnetic signature of the chip. One important aspect is the cost of such attacks. This type of attack is much cheaper than a reverse engineering attack which requires an electronic microscope to study the structure. Temporal analysis or timing attack [24] is another way to obtain cipher keys. The temporal reaction of the system leaks information enables the attacker to extract the cipher key or other secret information such as a password. Like with DPA, the attacker has to make certain assumptions about the information to be extracted, e.g. knowledge of the algorithm, in which case the branch instructions in the program can also help to solve a secret since a timing model of the algorithm can be established. Indeed, timing hypotheses are possible as the program running on the target device is often known. In this case, thanks to statistical studies, information can be extracted.

Fault injection [26] is the last way to obtain secrets through a side channel. However, like reverse engineering, more equipment is required than for the types of attacks described above. The injection of a fault into a system through the memory, for example, corresponds to a modification of a bit (laser or electromagnetic waves). Knowledge of the implementation of the algorithm is crucial to solve a secret. In most cases, the fault is inserted in the last round of an algorithm [26]. This is because the trace of the fault is more visible in the ciphered result. The goal of such hardware attacks is to obtain secret information from the chip. Regular improvements in side-channel attacks have been made in the last ten years and use advanced techniques to extract information. This book provides extensive discussions on the subject.

Denial of service attacks are different and the aim here is to cause a system breakdown. In autonomous embedded systems, power is an essential concern. It is one of the most important constraints on the system. To give an example, with a cell phone or a PDA, an attacker can perform a large number of requests that aim to activate the battery and hence to reduce the life of the system [27, 31]. In wireless communication systems, another type of attack activates the transmitter antenna to obtain the same result (i.e. reducing the life of the system). Increasing the workload of a processor is another way of consuming more battery. Indeed the workload of the system is related to power consumption, so an attacker may try to force the processor to work harder [27, 31]. As a consequence, the lifetime will be affected.

The range of attacks against a system is wide and depends on several parameters: the goal, the budget, and the type system concerned. Hardware attacks are already a serious threat to embedded systems but software attacks are becoming more and more dangerous and need to be recognized and prevented.

2.2.2 Software Attacks

Like in servers and workstations, embedded systems are being increasingly affected by viruses and worms [9]. The difference between a virus and a worm is that a virus requires the help of a human to infect a system and then to spread, whereas a worm does not. A worm is considered to be autonomous. All computer science concepts can be transposed to the embedded system domain. The replacement of a program by a malicious threatens the security of the system. The malicious program may either try to access sensitive data or to shut down the system. Concerning secret data, cipher keys are the most sensitive data as once the attacker knows the cipher keys, he/she has access to all the information in plain. Encrypting memory and protecting cipher keys are classical solutions to these attacks. However protections used in computer science are not appropriate for embedded systems (less computing power and memory). As a result, dedicated solutions for embedded systems are gradually emerging (e.g. bus or program monitoring) [8]. The number of attacks targeting embedded systems is also increasing rapidly. For example, a virus or a worm can be sent to the same system several times to launch the antivirus. Scanning the whole system increases the workload of the processor and thus decreases the battery lifetime which may be critical for autonomous systems. The concept of embedded systems extends the scope of viruses and of worms.

The classification of hardware and software attacks (as depicted in Fig. 2.1) is generic and can be applied to different platforms. Here we focus on attacks against FPGA-based designs which are more hardware oriented, as we explain in the following sections.

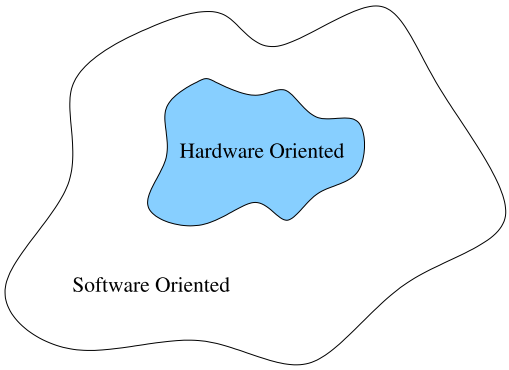

Fig. 2.1 Hardware and software attacks coverage against embedded systems

2.3 Objective of an Attacker

The most common threat against an implementation of a cryptographic algorithm is obtaining a confidential cryptographic key, that is, either a symmetric key or the private key of an asymmetric algorithm. Given that in most commercial applications, the algorithms used are public knowledge, obtaining the key will enable the attacker to decrypt future communications (assuming the attack has not been detected and countermeasures have not been taken) and, which is often more dangerous, to decrypt past communications that were encrypted.

Another threat is the one-to-one copy, or cloning of a cryptographic algorithm together with its key. In some cases this is enough to run the cloned application in decryption mode to decipher past and future communications. In other cases, execution of a particular cryptographic operation with a presumably secret key is—in most applications—the only criterion used to authenticate a party to a communication. An attacker who can perform the same function can attack the system.

Yet another threat is applications in which the cryptographic algorithms are proprietary. Even though such an approach is not common, it is a standard practice in applications such as pay-TV and in government communications. In such scenarios, it is advantageous for an attacker to reverse-engineer the encryption algorithm itself. The associated key might be recovered later by other means (bribery or classical cryptanalysis, for instance). The above discussion generally assumes that an attacker has physical access to the encryption device. Whether that is the case or not depends to a great extent on the application concerned. However, we believe that in many scenarios such access can be taken for granted, either through outsiders or through dishonest insiders.

2.3.1 Security System Using FPGAs

Based on reports by Wollinger et al. [50] and Wollinger and Paar [49], we list the potential advantages of FPGAs in cryptographic applications.

- *Algorithm agility*. This term refers to cryptographic algorithms switching during operation of the targeted application. While algorithm agility is costly with traditional hardware, FPGA can be reprogrammed on the fly.
- *Algorithm upload*. Upgrading fielded devices is conceivable with a new encryption algorithm. FPGA-equipped encryption devices can upload the new configuration code.
- *Architecture efficiency*. In certain cases hardware architecture can be much more efficient if it is designed for a specific set of parameters. One example of a parameter for cryptographic algorithms is the key. FPGA allows this type of architecture, and enables optimization using a specific set of parameters. Depending on the type of FPGA, the application can be completely or partially modified.
- *Resource efficiency*. The majority of security protocols are hybrid protocols that require several algorithms. As they are not used simultaneously, the same FPGA device can be used for both through runtime reconfiguration.
- *Algorithm modification*. There are applications that require modification of standardized cryptographic algorithms.
- *Throughput*. General purpose microprocessors are not optimized for rapid execution. Although typically slower than ASIC implementations, FPGA implementations have the potential to run much faster than software implementations (as with a processor).
- *Cost efficiency*. There are two cost factors that have to be taken into consideration when analyzing the cost efficiency of FPGAs: the cost of development and the unit price. The cost of developing an FPGA implementation for a given algorithm is much lower than for an ASIC implementation. Unit prices are not high compared with the cost of development. However, for high-volume applications (more than one million circuits) ASIC is usually the most cost-efficient choice.

FPGAs obviously have some interesting features and should not be discarded for security applications. To analyze the problem of FPGA security, it is important to define the model of computation to be used and the areas to be protected. There are three possibilities: The first is considering the FPGA and its surrounding (normally a processor and memory) as a trusted area; the second is restricting the trusted area to the FPGA itself; and the third is when certain functional parts inside the FPGA are considered to be trustworthy and others are not.

2.3.1.1 FPGA Based Security Models

In the context in which the FPGA and its environment is considered as a trusted area (Fig. 2.2), the main element involved in the system security is the I/O interface. In this case, the data entering or leaving the system need to be protected (for example, confidentiality and authentication). In fact three interfaces can be used depending on the boundary of the security perimeter. The first is related to the I/Os of the system. In a secure execution context no information should leak from this interface so all the data has to be encrypted. The second is related to the FPGA configuration file if remote configuration is possible. In this case, it is essential to protect the

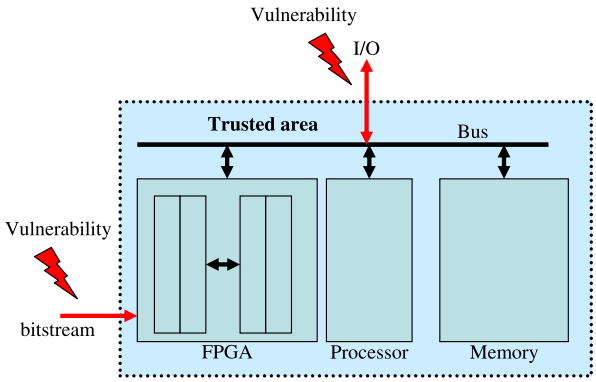

Fig. 2.2 FPGA, processor and memory trusted area: model one for FPGA-based secure systems

bitstream by encrypting it. If all the configurations are within the trusted area, they can be in clear form. The third interface is related to all the critical information dealing with the security of the system, for example the transfer of a new key or a new certificate. In this case, the interface needs to be different from the I/O one in order to increase the security of the system; this interface should also be protected. Generally authentication mechanisms are needed to ensure that only an authorized party can send new data through this interface.

In the second context, the FPGA is considered as a trusted area but its environment is not (Fig. 2.3). For such a model, in addition to protecting the I/O interfaces with the system as a whole, it is necessary to protect the communication inside the system in a per-block (Memory, Processor, FPGA) granularity. In this case, several techniques need to be considered in order to provide authentication, confidentiality and integrity verification. Furthermore, as the number of communications over the bus is generally critical, very efficient and optimized cryptography resources

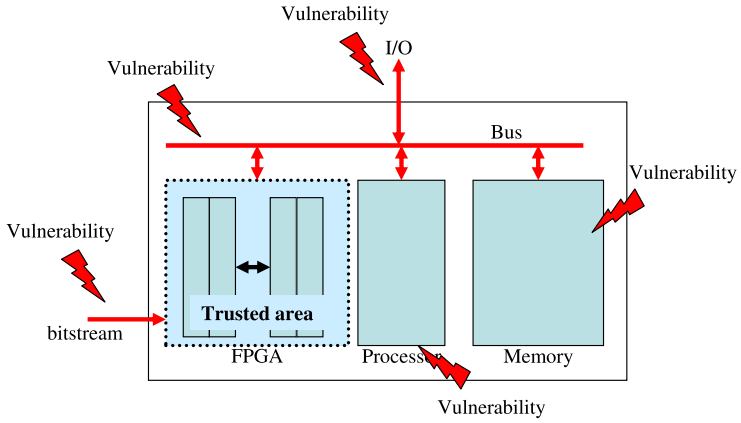

Fig. 2.3 FPGA trusted area: model two for FPGA-based secure systems

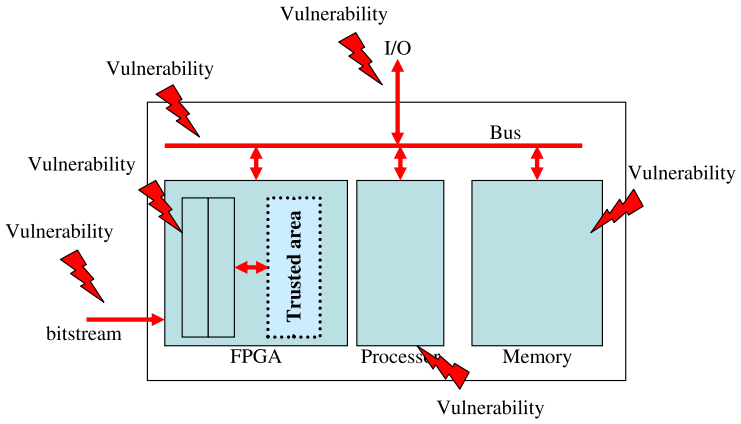

Fig. 2.4 Modules of FPGA trusted area: model three for FPGA-based secure systems

are needed. The latency of the exchanges is of paramount importance and needs to be tackled. In the second context, protecting the FPGA configuration is also an important issue.

Finally, in the third context, the FPGA itself contains regions that are trusted and regions that are not (Fig. 2.4). In such a situation, only the configurations (i.e. bitstreams) that compose the FPGA functionality need to be encrypted. FPGA's trusted area needs to be protected, but at this granularity, fine aspects of bitstreams have to be considered. This solution involves the same challenges as the previous ones since data exchanged within the FPGA but also exchanged with external resources need to be protected, but the solutions should lead to a very low overhead so as to not penalize the execution of the whole system. This model is clearly the most challenging one to design.

2.3.1.2 Threats Against FPGAs

Before building a solution it is essential for designers to clearly define the execution context and the kinds of threats they will be facing.

In this section, we describe several types of attacks against FPGAs. However, some of them will be discussed in detail in subsequent sections when we deal with FPGAs' vulnerabilities and countermeasures.

Black Box Attack—In this type of attack, the intruder sends all possible input combinations and with the results he/she obtains, he/she may be able to reverse-engineer a chip. In practice, this type of attack is difficult to perform on complex systems.

Read-Back Attack—Read back attacks are based on the ability to read the FPGA configuration, usually using the JTAG plug. This feature is provided in most FPGAs to promote debugging capabilities. The aim of the attack is to read the configuration

of the FPGA through the JTAG or programming interface to obtain secret information. Recently FPGAs vendors have considerably improved their devices to increase the level of protection.

Cloning of SRAM FPGAs—SRAM FPGA based systems normally store the configuration file in a non-volatile memory outside the FPGA. In such a situation, an eavesdropper can easily retrieve the configuration file flowing through the port, and possibly clone the same design in other FPGAs. The only possible way to protect the system in this case is to encrypt the bitstream. FPGAs vendors provide this possibility in their most recent devices.

Physical Attack against SRAM FPGAs—The goal of an attack targeting the physical layer of an FPGA is to investigate the chip design in order to obtain secret information by probing points inside the FPGA. These attacks target the parts of the FPGA that are not available through the normal I/O pins. Using instruments based on focused ions (FIB), for example, the attackers can inspect the FPGA structure and retrieve the design or the keys. Such attacks are hard to implement due to the complexity of the equipment required. Moreover, some technologies, like Antifuse FPGAs and Flash FPGAs, which have their own limitations, can make attacking even harder.

Side-Channel Attacks—In this case, the physical implementation of the systems is used to leak information like energy consumption, execution time and electromagnetic fields. By observing these phenomena, an attacker can obtain the power, time and/or electromagnetic signatures of the system, which, in turn, can reveal secrets concerning the underlying implementation. Gathering such signatures is one step in the problem. In fact, the data obtained still has to be processed to obtain the desired results. Very sophisticated techniques have been developed in the last few years that require few measurements to attack a system.

Data analysis—In fact, the data acquired by read back attacks, as well as those from side channel attacks, are considered as noise. The fact that an attacker possesses this information does not imply that he/she will be able to obtain the original design running in the FPGA, but nevertheless makes this possible.

Reverse-engineering is the work done after the bitstream has been obtained, for example, when it is necessary to discover the data structure used by the manufacturer to codify the configuration of the FPGA. Reverse engineering is not limited to bitstreams but can also be achieved by observing bus activities during program execution in a softcore processor implemented in the FPGA. Many reverse engineering attempts on FPGAs succeed, and normally the manufacturers use a disclosure term to morally refrain the attackers, which is not sure at all.

Leakage data is processed using techniques like *Simple Power Analysis* (SPA) or *Differential Power Analysis* (DPA). Normally the aim of these approaches is to identify energy consumption patterns similar to the ones obtained in the known execution pattern of a cryptographic algorithm. Then, finding the key is only a question of time and of the quantity of statistics obtained.

2.4 Security Requirements for Modules

The security of systems used to protect sensitive information is provided by cryptographic modules. Security requirements for cryptographic modules are specified in the Federal Information Processing Standard Publication (FIPS PUB 140-2) [33]. These requirements are related to the secure design and implementation of a cryptographic module.

2.4.1 Security Objectives

Security requirements for cryptographic modules are derived from the following high-level functional security objectives:

- To employ and correctly implement the approved security functions for the protection of sensitive information.
- To protect a cryptographic module from unauthorized operation or use.
- To prevent the unauthorized disclosure of the contents of the cryptographic module, including plaintext cryptographic keys and other critical security parameters (CSPs).
- To prevent the unauthorized and undetected modification of the cryptographic module and cryptographic algorithms, including the unauthorized modification, substitution, insertion, and deletion of cryptographic keys and CSPs.
- To provide indications of the operational state of the cryptographic module.
- To ensure that the cryptographic module performs properly when operating in an approved mode of operation.
- To detect errors in the operation of the cryptographic module and to prevent the compromise of sensitive data and CSPs resulting from these errors.

While the security requirements specified in the FIPS 140-2 standard are intended to maintain the security provided by a cryptographic module, conforming with this standard is necessary but is not sufficient to ensure that a particular module is secure. The operator of a cryptographic module is responsible for ensuring that the security provided by the module is sufficient and acceptable to the owner of the information that is being protected, and that any residual risk is acknowledged and accepted.

Similarly, the use of a validated cryptographic module in a computer or telecommunication system is not sufficient to ensure the security of the whole system. The overall security level of a cryptographic module must provide the level of security that is appropriate for the security requirements of the application and of the environment in which the module has to be used, and for the security services that the module has to provide.

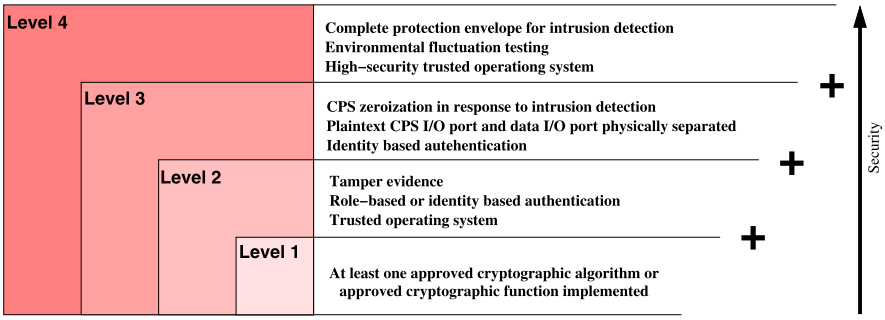

Fig. 2.5 Security requirements in four security levels, according to the FIPS 140-2 Standard

2.4.2 Security Levels

The standard defines four qualitative levels of security—Level 1 to Level 4 (Fig. 2.5). These four increasing levels of security enable cost effective solutions that are appropriate for different degrees of data sensitivity and different application environments.

Security Level 1 requires only the use of an approved algorithm or security function. It allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an unevaluated operating system. No specific physical security mechanisms are required in a Security Level 1 cryptographic module beyond the basic requirement for production-grade components. *Security Level 2* enhances the physical security of the module by adding the requirement for tamper evidence and a role based authentication of an operator. A trusted (verified) operating system with an approved degree of security should be used starting from Level 2. In *Security Level 3*, the security mechanisms should detect and respond to attempts at physical access, unauthorized use or modification of the cryptographic module. The tamper detection circuitry should "zeroize" all plaintext CSPs when a physical intrusion is detected. At this security level, the role based authentication from Level 2 is replaced by identity based authentication. At Level 3, the I/Os of plaintext CSPs should be performed using ports that are physically separated from other data ports. *Security Level 4* is the highest security level. The module should be enclosed in a complete protection envelope. Intrusion into the module should have a very high probability of being detected and the module should detect fluctuations in environmental conditions, namely voltage and temperatures that are outside the normal operating range.

2.4.3 Security Requirements

The security requirements specified in the FIPS 140-2 standard cover 11 areas related to the design and implementation of a cryptographic module. These

areas include cryptographic module specifications; module ports and interfaces; roles, services, and authentication; finite state models; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self tests; and design assurance. The last area concerned with the mitigation of other attacks has not yet been tested but the vendor is required to document implemented controls (e.g. differential power analysis). While most of these areas are related to the technology used (FPGAs in our case), some only concern design methodology (e.g. finite state models). Below, we analyze only those design and implementation areas that are directly related to the application of FPGAs in security modules. These areas include:

- Cryptographic module specification
- Cryptographic module ports and interfaces
- Physical security
- Operational environment
- Cryptographic key management
- EMI/EMC
- Self-tests

Cryptographic module specification deals with a combination of hardware, software and firmware means used to implement cryptographic functions and protocols within a defined cryptographic boundary. A cryptographic module should implement at least one approved security function used in an approved mode of operation. A cryptographic boundary should consist of an explicitly defined perimeter that establishes the physical bounds of a cryptographic module. If a cryptographic module consists of software or firmware components, the cryptographic boundary should contain the processor(s) and other hardware components that store and protect the software and firmware components. Hardware, software, and firmware components of a cryptographic module can be excluded from the security requirements if it is demonstrated that these components do not affect the security of the module.

Cryptographic module ports and interfaces—Two hierarchical levels of I/O ports are defined by the standard: physical ports and logical interfaces. The module should have at least four logical interfaces (data input, data output, control input and status output) that can share the same physical port. Data input and output are used to transfer plaintext and ciphertext data and plaintext (only up to Security Level 2) and ciphertext CSP. Starting from Security Level 3, the I/O port for plaintext CSP should be physically separate from other data ports. All data output via the data output interface should be prevented when an error state exists and during self tests. The control input interface serves to provide commands and signals, and to control data (including function calls and manual controls such as switches, buttons, and keyboards). The status output interface enables the design to send signals, indicators, and status data (including return codes and physical indicators such as Light Emitting Diodes and displays).

Physical security—A cryptographic module is protected by physical security mechanisms in order to restrict unauthorized physical access to the contents of the module and to avoid unauthorized use or modification of the module (including

substitution of the entire module) when installed. All hardware, software, firmware, and data components within the cryptographic boundary should be protected. Physical security requirements are specified for three defined physical embodiments of a cryptographic module:

- *Single-chip cryptographic modules* use a single integrated circuit (IC) chip as a stand alone device. Examples of single chip cryptographic modules include smart cards with a single IC chip. Although single chip cryptographic modules are the most vulnerable to side channel attacks, they are usually used in a hostile environment. Single chip modules based on FPGAs have not been used up to now, since they involve the use of a non-volatile technology.
- *Multi-chip embedded cryptographic modules* are physical devices in which two or more IC chips are interconnected. If possible, they should be embedded in an opaque enclosure. Examples of multi-chip embedded cryptographic modules include adapters and expansion boards.
- *Multi-chip stand-alone cryptographic modules* are physical embodiments in which two or more IC chips are interconnected and the entire enclosure is physically protected. Examples of multi-chip, stand alone cryptographic modules include encrypting routers or secure radios.

Operational environment—The operational environment of a cryptographic module refers to the management of the software, firmware, and/or hardware components required for the module to operate. The operational environment can be non-modifiable (e.g. firmware contained in ROM, or software contained in a computer with I/O devices disabled), or modifiable (e.g. firmware contained in RAM or software executed by a general purpose computer). An operating system is an important component of the operating environment of a cryptographic module.

Cryptographic key management—The security requirements for cryptographic key management encompass the entire life cycle of cryptographic keys, cryptographic key components, and CSPs used by the cryptographic module. Key management includes random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization. Secret keys, private keys, and within the cryptographic module, CSPs should be protected from unauthorized disclosure, modification, and substitution. Inside the cryptographic module, public keys should be protected against unauthorized modification and substitution. A cryptographic module may use random number generators (RNGs) to generate cryptographic keys and other CSPs internally. There are two basic classes of generators: deterministic and nondeterministic. A deterministic RNG consists of an algorithm that produces a sequence of bits from an initial value called a seed. A nondeterministic RNG produces output that depends on some unpredictable physical source that is beyond human control. There are no FIPS Approved nondeterministic random number generators. If intermediate key generation values are sent outside the cryptographic module, the values should be sent either (1) in encrypted form or (2) under split knowledge procedures. Key establishment can be performed by automated methods (e.g. use of a public key algorithm), manual methods (use of a manually-transported key loading device), or a combination of automated and

manual methods. Cryptographic keys stored inside a cryptographic module should be stored either in plaintext or encrypted. Plaintext secret and private keys should not be accessible to unauthorized operators from outside the cryptographic module. A cryptographic module should associate a cryptographic key (secret, private, or public) stored inside the module with the correct entity (e.g. the person, group, or process) to which the key is assigned. A cryptographic module should provide methods to zeroize all plaintext secret and private cryptographic keys and CSPs inside the module. Zeroization of encrypted cryptographic keys and CSPs or keys otherwise physically or logically protected inside an additional embedded validated module (meeting the requirements of this standard) is not required.

Electromagnetic Interference Compatibility (EMI/EMC)—Cryptographic modules should meet EMI/EMC requirements. While the metallic enclosure of the multichip cryptographic module enables these requirements to be met quite easily, standard single chip modules are difficult to design.

Self-Tests—A cryptographic module should perform power up self tests and conditional self tests to ensure that the module is functioning properly. Power up self tests should be performed when the cryptographic module is powered up. Conditional self tests should be performed when an applicable security function or operation is invoked (i.e. security functions for which self tests are required). If a cryptographic module fails a self test, the module should enter an error state and output an error indicator via the status output interface. The cryptographic module should not perform any cryptographic operations while in an error state. All data output should be inhibited when an error state exists. A cryptographic module should perform the following power up tests: cryptographic algorithm test, software/firmware integrity test, and critical functions test. It can also perform the following conditional self tests: pair-wise consistency test, software/firmware load test, manual key entry test, continuous random number generator test, and bypass test.

2.4.4 Security Policy

A cryptographic module security policy should consist of a specification of the security rules, under which a cryptographic module should operate, including the security rules derived from the requirements of the standard and the additional security rules imposed by the vendor. There are three major reasons that a security policy is required:

- It is required for FIPS 140-2 validation.
- It allows individuals and organizations to determine whether the cryptographic module, as implemented, satisfies the stated security policy.
- It describes the capabilities, protection, and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it meets their security requirements.

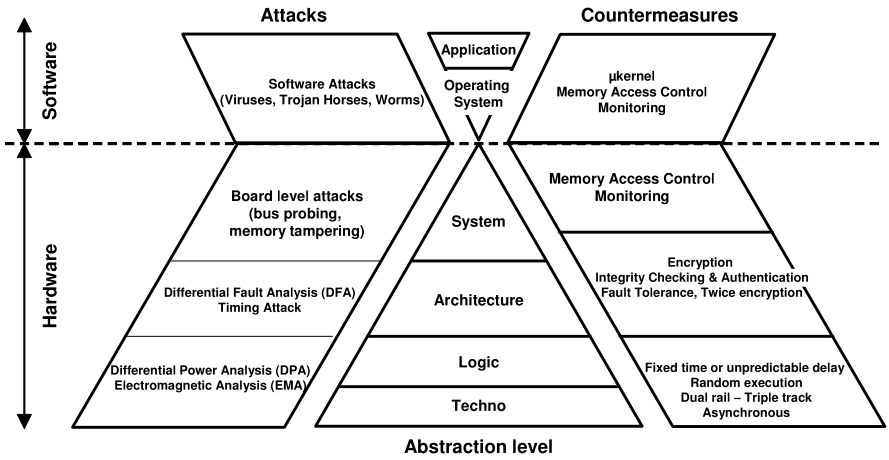

Fig. 2.6 Security pyramid: toward a defense-in-depth

2.5 Vulnerabilities of FPGAs

When dealing with security, it is important for the designer to not disregard any parts of the security barriers. The strength of a system is defined by its weakest point; there is no reason to enhance other means of protection if the weakest point remains untreated. To address this fundamental issue, the different hierarchical levels of a design (from application to technological levels) must be reviewed. Each level has specific hardware or software weaknesses, so specific mechanisms need to be defined in order to build a global secure system (i.e. defense in depth). Depending on the requirements and the security to be reached, several levels have to be considered. Thus it is important to clearly define the security boundaries for the system to be protected. In this chapter, we tackle this point by defining the security pyramid that covers the levels of security. In the following sections we address the different levels from technological to system levels as highlighted in Fig. 2.6 and slightly discussed into the Chap. 1. Operating system and application levels are beyond the scope of this chapter as we only deal with hardware solutions. However it is important to bear in mind that they need to be taken into consideration for a complete system. A lot of threats have to be considered, all information that leaks from a cryptographic device can be exploited by an attacker. For example, attacks based on power consumption or electromagnetic emission can be performed with low competencies and at low cost. Furthermore, attackers can cause errors during the encryption (or decryption) process aimed at to obtaining secret information, such as cryptographic keys. Therefore cryptographic devices must be protected against fault injection and leakage information. Many countermeasures are now well established for ASIC, but this is not yet the case for FPGA devices. Only a few academic or industrial studies have been conducted to ensure the confidentiality and integrity of FPGA devices. Nevertheless, this topic has attracted a lot of attention in the last few years and major progress is underway.

2.5.1 Technological Level

The technological level corresponds to the device and mainly concerns tamper evidence or resistance. Many authors have targeted the technological level but mainly for ASIC-based designs. However in [50] and [49], the authors performed an in-depth analysis of attacks against FPGAs at the technological level. Several technologies are possible for FPGAs, the most widespread being SRAM, Flash and Antifuse. Each technology has advantages but also some limits. Secured ASICs frequently incorporate mechanisms able to detect an attempt by an attacker to make an invasive attack, for example by removing package sealing and by inserting probes at appropriate points in order to obtain cipher keys. At chip level, the mechanism could be distributed sensors to check the package has not been removed. These secured packages are also suitable for FPGA devices. However at die level, FPGA users are limited by the capabilities of the device, they cannot add special security mechanisms such as analog sensors. The level of protection provided by FPGAs technologies is an interesting metric to identify the studies required to improve the security level. In the IBM Systems Journal, Abraham et al. [1] defined the security levels for modern electronic systems.

- *Level 0 (ZERO)*—No special security features added to the system. It is easy to compromise the system with low cost tools.
- *Level 1 (LOW)*—Some security features in place. They are relatively easily defeated with common laboratory or shop tools.
- *Level 2 (MODLOW)*—The system has some security against non-invasive attacks; it is protected against some invasive attacks. More expensive tools are required than for level 1, as well as specialized knowledge.
- *Level 3 (MOD)*—The system has some security against non-invasive and invasive attacks. Special tools and equipments are required, as well as some special skills and knowledge. The attack may be time consuming but will eventually succeed.
- *Level 4 (MODH)*—The system has strong security against attacks. Equipment is available but is expensive to buy and operate. Special skills and knowledge are required to use the equipment for an attack. More than one operation may be required so that several adversaries with complementary skills would have to work on the attack sequence. The attack could fail.
- *Level 5 (HIGH)*—The security features are very strong. All known attacks have failed. Some research by a team of specialists is necessary. Highly specialized equipment is necessary, some of which might have to be specially designed and built. The success of the attack is uncertain.

According to this classification, it is possible to specify a general security level for existing FPGA technologies and ASIC circuits. These levels are not fixed and depend on the factory and the type of circuit (several families may be processed in the same factory and some of them may be highly security efficient, like military families). Table 2.2 lists the security levels of existing technologies, especially FPGA circuits.

Table 2.2 Security level of classical integrated circuits

Integrated circuit	Security level
SRAM FPGA	0
ASIC gate array	3
Cell-based ASIC	3
SRAM FPGA with bitstream encryption	3
Flash FPGA	4
Antifuse FPGA	4

SRAM FPGAs need a bitstream transfer from the root EEPROM at power up (due to the configuration memory that is an SRAM volatile memory). Consequently, it is easy for a hacker to read the bitstream during the transfer using a simple probe.

Thus unprotected SRAM FPGAs are not efficient for safe design. However, with a bitstream encryption it is possible to considerably improve the security level since the security weakness is overcome. SRAM FPGAs have a good resistance against some attacks like power analysis. Although ASICs are often considered to be a secure technology, they are actually relatively easy to reverse engineer. Because, unlike FPGAs, ASICs do not have switches, and it is thus possible to strip the chip and copy the complete layout to understand how it works. Methods to reverse engineer ASIC exist. The cost of reverse engineering is high since the tools required are expensive and the process is time consuming. It is not a simple process and the security level is 3 for such devices. Contrary to ASICs, FPGAs, like antifuse or flash, are security efficient since they are based on switches. With these FPGAs, no bitstream can be intercepted in the field (no bitstream transfer, no external configuration device). In the case of antifuse FPGAs, the attacker needs a scanning electron microscope to identify the state of each antifuse. Nevertheless, the difference between a programming and a non-programming antifuse is very difficult to see. Moreover, such analysis is intractable in a device like Actel AX2000 that contains 53 million antifuses and, according to Actel (http://www.actel.com/products/solutions/security/), only 25% (on average) of these antifuses are programmed. For flash FPGAs, there is no optical difference after configuration, so invasive attacks are very complex. The same advantages are cited by QuickLogic to promote their flash FPGAs with ViaLink technology. Even if the antifuse and the flash FPGAs are very security efficient, they can only be configured (or programmed) once, so they are not reconfigurable devices. Furthermore as defined by the FIPS 140-2 standard, for security levels 3 and 4, it has to be possible to zeroize all the secret information, which is not possible with technologies like flash and antifuse. The system built with these devices is thus not flexible. If the designer wants a reconfigurable device, he should choose an SRAM FPGA. Moreover, the capacities of the SRAM FPGAs are the highest for FPGA devices. The market share of SRAM FPGAs is more than 60% (just counting the two leading companies Xilinx (http://www.xilinx.com) and Altera (http://www.altera.com)). Further research is required to improve the security level of such FPGAs and particularly to improve bitstream encryption. In recent years, FPGAs vendors have been tackling this point

in order to provide their customers with efficient solutions to encrypt the SRAM FPGA bitstream. However, they still have some drawbacks and could be further improved by taking the latest innovations of these FPGAs into account.

2.5.2 Logical Level

It is now widely recognized that the Achilles' heel of secure applications, such as 3DES and AES ciphering algorithms, is their physical implementation. Among all the potential techniques to retrieve the secret key, side channel attacks are worth mentioning. Although there are a lot of different types of side channel attacks, the DPA attack is considered to be one of the most efficient and most dangerous since it requires few skills and little equipment to be succeed. Thus at the logical level, all attacks that are possible on ASICs, like side channel attacks and fault attacks, are reproducible in FPGAs. Secret leakage can even be amplified, which makes the attack easier. At first sight, FPGAs appear to be less robust than ASICs. This could have different causes:

- The FPGA structure has heavy loaded wires made up of long lines or lines segmented by pass transistors. As power consumption is proportional to the capacitance load, this makes measuring power consumption more easy.
- The designers often take advantage of pipelining in FPGAs, as the DFF (D Flip Flop) is "free" in every cell. The DFF is not only used as a power contributor to power consumption in CMOS ASICs but as predictor (for correlation attacks) of relevant key dependent consumption for the attack strategies. The FPGAs have very rapid (for performances and fighting metastability) and high power consuming DFFs at their disposal but they also drive logic that could be predictable.
- The use of pass transistors generates a power consumption expression that varies partly in Vdd^3 and not only in Vdd^2, and as the CMOS gates are just under the conduction threshold, this helps make measuring the power consumption easier in FPGAs.

All these reasons make FPGAs more vulnerable to attacks since they are based on variations in power (analyzed for passive attacks and provoked for fault attacks) on cells where the key is involved. When analyzing power consumption during a ciphering operation, peaks are clearly discernible in the acquisition trace at every clock period. An efficient attack on FPGA is based on predicting the transitions of relevant DFF or logic driven by these DFFs, where there could be contribution to or leak from the key. For instance, it is possible to predict logic states at the S-box outputs during the first and last round of symmetric key algorithms. Many attacks on FPGA implementations have been reported in the last five years. For example, in [34], an SPA was successfully used in an unprotected implementation of an elliptic curve cryptographic algorithm. Furthermore as underlined in [34], attacks on FPGAs allow a hacker to:

- Evaluate the intrinsic resistance or vulnerability of this device class;

- Assess the resistance or vulnerability of the algorithms, executed on a real concurrent platform. Compared to a simulation, measurements made on an emulating device are indeed expected to be closer to the final projection in an ASIC.

In [42] and [39], the authors describe successful correlation power attacks (CPAs) against DES and AES implementations programmed into FPGA. The attacks scenario does not differ from that of ASICs (such as smart cards). Unsurprisingly, in all cases, the attacks were just as successful as when carried out against hardwired devices. As a consequence, there is a real incentive to devise countermeasures that take the architecture of FPGAs into account. It is important to come up with innovative solutions based on accurate knowledge of the FPGAs' internals in order to counter these attacks. Current FPGAs vendors have not yet taken up this challenge.

2.5.3 Architecture Level

Logical errors can be used to perform attacks at algorithmic level. A simple example is an attack against RSA algorithm implementation. RSA is based on modular exponentiation, decryption of a ciphered message C follows the following formula:

$$M = C^E \bmod N,$$

where C is the ciphered message, E the private exponent, N the public modulus, and M the unciphered message

The usual way to perform this calculus is the square and multiply algorithm, for either software or hardware implementation.

In the original algorithm, multiplication is carried out only if the exponent bit concerned is 1. So a dummy operation can be inserted in order to have a constant computation time. Supposing that an attacker can create an error at a precise algorithm step, he/she could easily guess the decryption key, which is secret information. To succeed, the attacker must create an error during the first multiplication and only the first; at the end of the algorithm, if the result is wrong, the multiplication was not a dummy one, so the first bit of the exponent is 1. If the result is correct, the multiplication was a dummy one, so the first bit of the exponent was 0. The other exponent bits can be discovered by repeating this scheme for all the multiplication steps. There are many ways to create an error during a computation. A simple way is to increase clock frequency above the maximum allowed by the attacked device. Uncommon temperature, voltage supply or clock glitch can also cause computation errors. With laser or focus ion beams, errors can be created in specific areas in the circuit. This type of fault injection requires specific expensive equipment but can provide better results. In the previous example, attackers do better by introducing errors only in the multiplier area, consequently other operations will not be affected. Other types of fault analysis exist, [16] deals with such attacks on smart card implementation of AES symmetric cipher. Of course this attack can target ASIC as well as FPGA platforms. However, in the future it will be interesting to see if FPGA structure could provide specific countermeasures. This point was judiciously exploited

by [39] to explore the advantages and limits of pipelining techniques from a security point of view. This architectural aspect (along with scheduling and resources allocation) is indeed a new dimension when refining a software code into an RTL description. FPGAs are appropriate tools for the rapid comparison of different architectures. At the architectural level, one basic block to consider is the processor. In the following section an example is given based on 32-bit processor and results obtained with SCA.

2.5.3.1 Processor Level

The MicroBlaze and the Nios II are 32-bit soft-core processors designed and supported by Xilinx and Altera respectively, for their FPGAs. Because of the growing number of embedded systems involving applications with security services, it is necessary to analyze potential weaknesses of these architectures used in most of the modern FPGAs.

The MicroBlaze and Nios II implement both a classic RISC Harvard architecture exploiting the Instruction Level Parallelism (ILP) with a 5-stage pipeline: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MA), Write Back (WB). All pipe stages are hooked together with a set of registers, commonly named pipeline registers. They play the main role of carrying data and control signals for a given instruction from one stage to another.

As a case study, an evaluation was conducted on the MicroBlaze with the Data Encryption Standard. The processor was programmed in ANSI C code on a Xilinx Spartan-3 Starter Kit board (XC3S1000 FPGA). Without loss of generality, the attacks were performed at the end of the first encryption round, when the left part of the plaintext message and the result of the Feistel function are XORed. The result of this operation contains a partial information about the cipher key, and thus, is a potential point of attack.

Critical instructions are detailed in the following list:

- **LWI R4, R19, 44**; the result of the Feistel function is loaded from data memory into register R4
- **LWI R3, R19, 40**; the left part of the message is loaded from data memory into register R3
- **XOR R3, R4, R3**; the result of the XOR operation between R3 and R4 is stored into register R3
- **SWI R3, R19, 32**; the content of R3 is stored into data memory

According to these instructions and to the chosen model of attacks, the sensitive data are handled during the XOR and the SWI instructions. During the execution of these instructions (in Fig. 2.7), we observe that the sensitive data are unprotected during the EX, MA, and WB stages.

A Differential ElectroMagnetic Analysis (DEMA) was conducted on the proposed DES software implementation [6]. The secret key was discovered with less than 500 electromagnetic traces. Besides, the most interesting result is the effect

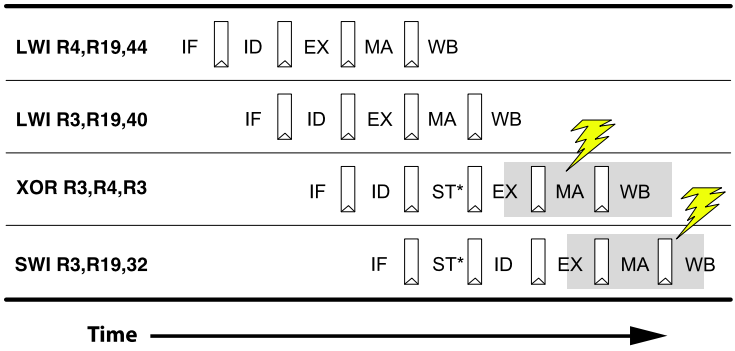

Fig. 2.7 The 5-stage pipeline during the execution of critical instructions

of the pipelining technique. Figure 2.8 illustrates the DEMA obtained for the first sub-key (50,000 electromagnetic traces were collected to emphasize the impact of this hardware feature). In the picture, the black curve refers to the correct sub-key, while the others correspond to the wrong sub-key hypotheses (the guessed sub-key is characterized by the highest amplitude).

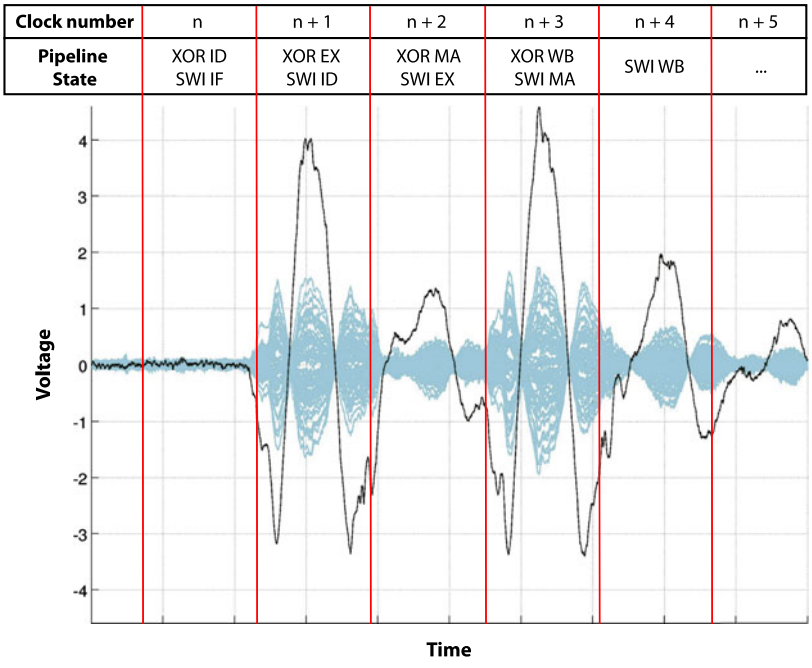

Fig. 2.8 DEMA traces obtained for the first sub-key of the DES software implementation with the MicroBlaze

The margin is basically defined as the minimal relative difference between the amplitude of the differential trace obtained for the guessed sub-key (black curve) and the amplitude of the differential traces obtained for the other sub-keys (other curves). This one reaches more than 50% for the correct sub-key during several time periods, which underlines the considerable vulnerability of a pipelined architecture.

The conclusion to be drawn from the above investigation seems fairly straightforward: the pipelined datapath of embedded microprocessors for FPGAs is a critical security issue.

2.5.4 System Level

At this level, we consider the system as a set of communicating blocks. Some are secure, and the main focus is communication between the modules. One of the threats at the system level is the "man in the middle" attack, in which an adversary eavesdrops on the communication channel. The security of the blocks depends on the cryptographic algorithm used to cipher the information. The problem is ensuring that the communication is secure in an unsecured channel. In addition, guaranteeing security has a cost and the cost of communication may be increased by the use of security dedicated modules integrated in the original design. The solutions described below are some of the recent techniques described to address this problem. Some of them are orthogonal in relation to others, meaning that we simultaneously can use a set of techniques. As mentioned in previous sections, in an FPGA context, the configuration bitstream is one potential weakness in secure applications. This bitstream can be used to perform various types of attacks and needs to be considered at the system level in order to define a global solution.

- *Bitstream retrieval*—The first step is to retrieve the entire bitstream from the system. The easiest way is to use the read-back capabilities of most FPGAs. This function, which is used for debugging, allows a bitstream to be extracted from an FPGA device during run time. Of course, in a secure context, this feature has to be disabled, but this is not sufficient, since in low cost SRAM FPGAs, bitstream retrieval is very simple, even without a read back mechanism. Indeed the bitstream is stored in an external EEPROM, so the attacker can probe the data line between the FPGA and EEPROM to obtain it. This threat does not exist for non-volatile FPGAs because configuration data are stored inside the device, so only intrusive attacks are possible. Bitstream encryption mechanisms are available for most advanced FPGAs. A secret encryption key is stored inside the programmable device, while for volatile FPGAs an external battery is used to maintain key value. Thus the device accepts an encrypted bitstream and uses its dedicated decryption engine to obtain unciphered data. Attackers cannot decrypt the bitstream without the secret key. With this feature, attackers have to discover the secret key they need to recover bitstream data through (for example) an intrusive attack.

- *Attackers' objectives concerning bitstreams*—When an attacker discovers the FPGA configuration, a lot of information, some of which may be critical, becomes accessible. The first threat is bitstream reverse engineering; some companies are specialized in FPGA reverse engineering [14]. In this way, attackers could potentially access all information stored in the FPGA architecture, possibly secret cipher keys or maybe the structure of cryptographic algorithms not available to the public. Attackers can also inject their own bitstream into the programmable device, in this way reverse engineering becomes unnecessary, since the attacker could simply create a dummy system. For example, if the FPGA is used to encrypt written data to a hard disk drive, the attacker could build a system that does not cipher data. Another threat is fault injection into the bitstream [11]. Even without knowledge of the architecture, small modifications can seriously modify the system. Fault attacks are also suitable in this context. The final drawback of FPGA is cloning. An attacker in possession of the configuration data can clone the device ignoring possible intellectual property rights. This is not a real security weakness, rather an industrial threat that is specific to programmable devices. In conclusion, the bitstream is the image of the underlying FPGA architecture, it is similar to ASIC layout, therefore configuration data have to be protected.
- *Remote configuration*—Remote configuration is an interesting FPGA feature that allows systems to be upgraded by removing a potential security breach, or by upgrading algorithms. But this feature must be extremely well secured since it gives many possibilities to attackers. The first related threat is undesired reconfiguration, i.e., the design could be changed remotely by the attacker without user permission. A simple switch button could avoid this threat. The second is "man in the middle" attacks. The user wants to upgrade his/her programmable security device, but an attacker intercepts the request and replies with a fake configuration. For that reason, the connection must be secured by an authentication and integrity checking engine. Such secured mechanisms are already suitable for most FPGAs, Actel or Xilinx FPGAs, and application notes [2] describe secured remote configuration schemes.

Key management is also a major concern for embedded systems. This threat is not specific to FPGAs, ASICs are also vulnerable. Cryptographic devices need to store some symmetric and asymmetric keys to perform encryption or decryption. During system use, these cryptographic keys need to be changed or generated randomly. So these types of devices require secure key management.

2.6 Countermeasures

In the previous section we described some vulnerabilities of FPGAs from the technological level up to the system level. Fortunately it is possible to deal with these vulnerabilities to take advantages of the flexibility and the performances offered by FPGAs. In this section, we first review technological issues and then present logical solutions, such as dual-rail or asynchronous techniques. At the architecture level,

we highlight the fact that the parallelism and flexibility provided by FPGAs can be very efficient aids in building a secure primitive. Finally at the system level, the main issue is related to communication security and monitoring of the behavior of the system. We describe several techniques to address these problems.

2.6.1 Technological Level

At the technological level, the main countermeasures are related to physical processes, circuits, and packaging [49, 50]. Some sensors can be used to detect any attacks against the device. With standard FPGAs, the user is blocked by chip capabilities as most FPGAs do not have analog parts that can monitor environmental parameters. In the latest Actel FPGA, called Fusion, various analog functions are available like temperature sensors, clock frequency or voltage monitoring. These functionalities can be used to detect possible fault injection. To prevent physical attacks, it is also necessary to make sure that the retention effects of the cells are as small as possible, so that an attacker cannot detect the status of the cells. The solution would be to invert the data stored periodically or to move the data around in memory. Cryptographic applications cause long-term retention effects in SRAM memory cells by repeatedly feeding data through the same circuit. One example is specialized hardware that always uses the same circuits to feed the secret key to the arithmetic unit. This effect can be neutralized by applying an opposite current or by inserting dummy cycles in the circuit. In terms of FPGA application, this is very costly and it may even be impractical to provide solutions like inverting the bits or changing the location of the whole configuration file. One possible alternative would be to change only the crucial part of the design, like the secret keys, through dynamic partial reconfiguration. Counter techniques such as dummy cycles and opposite current approach can also be used for FPGA applications. Technological issues depend on the FPGA vendor but solutions like dynamic reconfiguration can provide some opportunities to strengthen the security of the device against certain tampering techniques that benefit from the retention effect of cells. Using sensors inside FPGAs will certainly become more common in the future to monitor the internal activity of FPGAs.

2.6.2 Logical Level

So far, the published countermeasures against attacks on FPGAs have mainly been inspired by techniques used to protect ASICs. The “Masking” and “Hiding” methods are among the most efficient ways of protecting FPGAs. Hiding consists in making the power consumption as constant as possible by using dual rail with precharge logic (DPL) and a four-phase protocol inspired by Asynchronous Logic. This ensures that:

- Transitions are independent of the target values (logical 0 or 1), because the information is encoded on two indiscernible wires;
- No leaks are caused by consecutive data correlation because of the intermediate precharge phase.

The most straightforward countermeasure built on the DPL is referred to as "wave dynamic differential logic" WDDL [43, 44]. The four-phase cadence is enforced by appropriate registers. In the meantime, the differential logic uses positive dual functions. The WDDL logic could be more secure in FPGAs than in ASICs, because the two dual gates are implemented in LUTs, which ideally are much alike. However some flaws have been underlined. Vulnerabilities are caused by the imbalance between the two networks and above all by the "early evaluation" (EE) effect. The latter is due to the differences in delay between two gate inputs. The effect is particularly marked in FPGAs where there is a wide range of routing. Associated with the need for speed and a low complexity design, new DPLs have been devised to enhance WDDL in FPGAs, among which: MDPL [35], STTL [36], BCDL [32].

The masking countermeasure allows the mean of the power consumption to be balanced by masking the sensitive data with a random variable [18, 45, 46]. Hence the observation is theoretically not exploitable by an adversary. In FPGA technology, the masked data are computed at the same time as the mask itself. This implementation, called "zero-offset" [47], allows the designer to keep a high level of ciphering throughput. However, whatever the implementation (hardware or software), the masking protection could give rise to higher order attacks [30]. Hence enhanced masking structures should be designed and used in FPGAs. One example is provided in the chapter on countermeasures.

Other countermeasures are possible at the logical level:

- *Random underlying operations*—FPGAs enable many more attacks to be defeated than the fixed architectures implemented in ASIC devices. A promising solution is to frequently modify the operation used in any particular cipher. For example, S-Box transformation used in the most recent symmetric bloc cipher (AES) could be computed in different ways, e.g. using Galois field $GF(2^n)$ operations, using a big lookup table or using the internal RAM memory embedded in most FPGAs. In this way, the power consumption would differ for each type of S-Box implementation.
- *Random noise addition*—A lot of countermeasures have been proposed, from clock randomization, power consumption randomization, or compensation [11] to tamper detection. However, longer differential power analysis generally leads to recovery of the secret key. With enough samples, the statistical analysis used by DPA can remove any random noise on power consumption. To understand why, we need to understand that DPA attacks succeed because a calculus always leaves the same power consumption trace. Adding random noise to power lines is not enough, because it could be removed by a number of different signal processing techniques. This can be represented by:

$$TotalPower(t) = RealPower(t) + RandomPower(t),$$

where *TotalPower* is the power trace that an attacker can easily measure, *RealPower* is the power consumed by the cryptographic algorithm that the attacker wishes to obtain, and *RandomPower* is the random noise added to defeat DPA. To extract the *RealPower*, the attacker could collect a large number of *TotalPower* traces and then average the collection. With enough *TotalPower* traces, the average will be very close to *RealPower*. According to this analysis, adding random noise is not a perfect solution, it could make DPA longer but would not make it impossible. A better way is to act directly on the *RealPower* factor.

In the FPGA context, other types of methods can be applied using runtime reconfiguration. For example, in the case of the AES algorithm, differential power analysis targets sub-byte operations. Power consumption is generally correlated with data involved in the sub-byte function because this operation is always done with the same logic gates. With partial reconfiguration, the sub-byte function could be done differently at every computation, so power consumption would be different. This method is similar to masking mechanisms, the input data and the final result may be the same but the underlying operations are not identical, so that power consumption is no longer correlated with the data. These different points will be analyzed in detail in the following chapters of this book.

2.6.3 Architecture Level

At the architectural level, some solutions use a method called LRA (leak resistant arithmetic) based on a random number representation [4, 7]. In these works, input data are represented according to a residue number system base which can be selected randomly before or during the encryption process. After computation, the result is converted into a classical binary representation. To help the reader understand this approach, we will explain some mathematical concepts. Leak Resistant Arithmetic (LRA) is based on the Residue Number System and on Montgomery's modular multiplication algorithm proposed in [4]. The Residue Number System (RNS) relies on the Chinese Remainder Theorem (CRT). This theorem states that it is possible to represent a large integer using a set of smaller integers. A residue number system is defined by a set of k integer constants, $m_1, m_2, m_3, \ldots, m_k$, called moduli. The moduli must all be co-prime; no modulus can be a factor of any other. Let M be the product of all the m_i. Any arbitrary integer X smaller than M can be represented in the residue number system as a set of k smaller integers $x_1, x_2, x_3, \ldots, x_k$ with

$$x_i = X \bmod m_i.$$

With this representation, simple arithmetic operations, like additions, subtractions and multiplications, can be performed absolutely in parallel. Once represented in RNS, the operations do not generate any carry. For example, adding A and B in RNS is just:

$$Sum_i = (a_i + b_i) \bmod m_i.$$

This sum involves k modular additions that can be performed in parallel. However, the conversion from RNS to decimals is not so simple. Fortunately, this conversion is only needed after all modular multiplications, so its cost is amortized. For cryptographic applications, modular reduction ($x \bmod M$), modular multiplication ($x * y \bmod M$) and modular exponentiation ($x^y \bmod M$) are some of the most important operations. They can be calculated using Montgomery's algorithm, modified for RNS representation. In addition to possible parallelism, LRA could also provide an algorithmic countermeasure against side channel attacks. With LRA, it is possible to compute the same calculus (RSA or ECC algorithms) in different bases. The goal of this approach is to randomize intermediate results. The same modular multiplication leads to different basic RNS operations, so power consumption will differ if the RNS base differs. RNS bases have to be chosen randomly so the attacker cannot obtain the base value, and by extension, the underlying RNS operations.

Another issue is related to fault injection at the architectural level with the aim of recovering sensitive data. A common way to avoid fault injection attacks is to use redundancy [2]. Critical parts in the design are replicated, and then outputs are compared, which generally allows errors to be detected. Mathematical error detection can also be used, [7] shows how to use redundant information to detect potential errors during calculus. This mechanism also relies on RNS, but an extra modulus is used to check the correctness of the result. Verification is possible at the end of each modular multiplication, so this type of attack can be detected. Moreover the physical location of each operator can be changed dynamically during the lifetime of the FPGA. Therefore, accurate fault injection using laser or focused ion beams is no longer possible, as attackers cannot identify the exact operator position.

Agility is another important metric for cryptographic applications since, as mentioned previously, providing a moving target increases the complexity of setting up an attack. To illustrate this point, the following case study deals with an AES security primitive. All selected implementations were performed on Xilinx Virtex FPGA, which is a fine grain configurable architecture. For this architecture, the configuration memory relies on a 1D configuration array. This is a column based configuration array and so partial configuration can be performed only column by column. For security issues, this type of configuration memory does not provide full flexibility but still enables partial dynamic configuration to perform security scenarios. Figure 2.9 summarizes all the different implementations in four charts; each chart corresponds to specific parameters. Figure 2.9.a corresponds to the AES cryptographic core security primitive with BRAMs (i.e. embedded RAM) on non-feedback mode [15, 20, 28, 37, 41]. Thus key setup management is not used for these studies. Concerning agility, all the solutions are based on static and full configuration. The configuration is defined through predefined configuration data and performed remotely. The configuration time is on average tens of ms, since full configuration is performed. The security module controller is not addressed in these studies since the implementations are static. Figure 2.9.a shows that various area/throughput trade-offs are possible depending on the implementation. It is important to dynamically adapt the performance and to ensure the security of the module. From the security point of view, it enables the global system to behave as a moving target, while from the

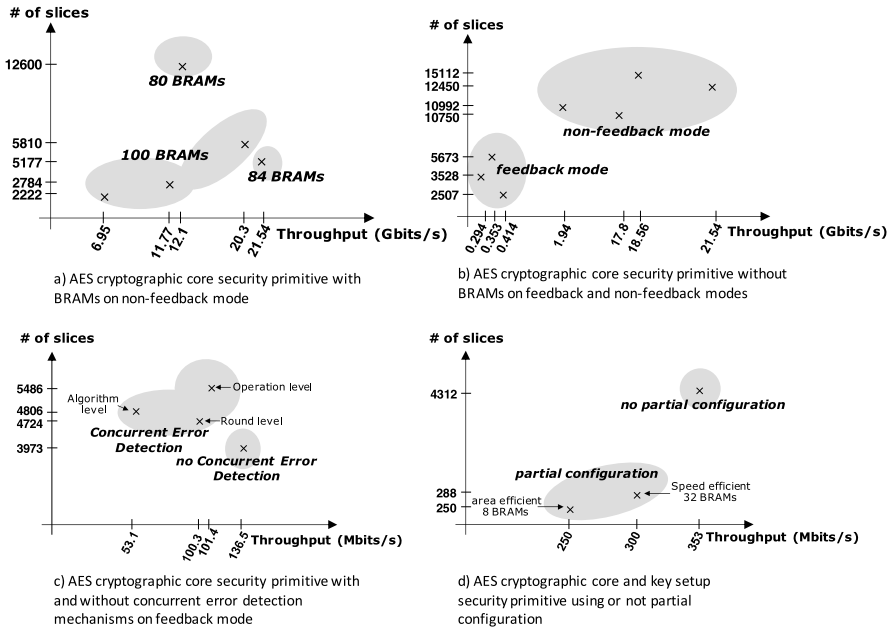

Fig. 2.9 Agility design space for the AES security primitive: throughput/area/reliability trade-offs

performance point of view, it allows different throughputs to be used dynamically depending on the actual requirements of the application. Figure 2.9.b corresponds to the AES cryptographic core security primitive without BRAMs on feedback [10, 13, 15] and non-feedback modes [13, 20, 22, 40]. Like in the previously example, key setup management is not used. Solutions [10, 13, 15] correspond to feedback mode while the others correspond to non-feedback mode. Feedback solutions enable throughput of on average hundreds of Mbits/s whereas non-feedback solutions enable around tens of Gbits/s. The same remarks as previously apply to agility characteristics; static, full and predefined configuration is used. In these studies the goal is to promote high throughput while reducing area and dealing with a specific execution mode. However as explained in this section, dynamism and reliability also have to be considered.

Figure 2.9.c shows different ways to manage fault detection that ensure reliability, an essential feature for security. Faults can be detected at different levels of granularity from algorithm to operation level [23]. The performance/reliability trade-off is interesting since a finer level of granularity enables reduced fault detection latency and facilitates a rapid reaction against an attack. But the price of this efficiency is area overhead. No type of error detection can improve performance, it is thus important to dynamically adapt the level of protection depending on the environment and on the state of the system. Concerning agility, static, full and predefined configuration is used. Finally, Fig. 2.9.d provides some interesting values since solutions using dynamic configuration are proposed. In [10], full configuration with predefined configuration data is implemented, whereas in [29] partial configuration

with dynamic configuration data is performed. In both cases, remote configuration is performed since the Configurable Security Module is considered be an agile hardware accelerator. Both solutions also deal with key setup management, in [10] this is performed inside the module so that the architecture is generic and in [29] it is performed by the remote processor, which means key specific architecture can be provided. In [29] the remote processor implements the security module controller that computes the new configuration when new keys have to be taken into account by the cryptography core. This type of execution enables considerable flexibility since the configuration data can be defined at run time. However in that case, the computation time to define the new configuration data is in the range of 63–153 ms, which may be prohibitive for some applications. The reconfiguration time for new configuration data is not critical (around tens of μs) since only partial configuration is performed. As can be seen in Fig. 2.9.d, partial configuration enables significant area savings compared to a generic implementation since in the latter, the architecture is specialized for each key. The security policy supported by the security module controllers are not explicitly presented in these works. Figure 2.9 shows that different solutions can be implemented for the same security primitive and so different area/throughput/reliability trade-offs can be considered. Agility enables these trade-offs and consequently both performance and security to be dynamically adapted to the actual execution context. The last important point is related to power consumption which has not been tackled in previous studies, even though for embedded systems, it may be an essential feature. In [38], energy efficient solutions are proposed for the AES security primitive. In this case, the important metric is Gbits/joule, which is very relevant since ambient systems are mobile.

It is important for designers who have to build modules to be aware of all these trade-offs in order to promote agility and to meet performance requirements. Further detailed studies on configuration power consumption, secure communication links and security module controller policy are required in order to propose secure modules and by extension, secure systems. However agility provides many keys to building high-security/high-performance systems.

2.6.4 System Level

Generating a perfectly secured system is an illusion; so the best way to prevent unanswerable threats is to detect them and to respond appropriately. Attacks against secured devices vary and many parameters have to be monitored to detect them. The SAFES (Security Architecture for Embedded Systems) approach focuses on protecting embedded systems by providing an architectural support for the prevention, detection and remediation of attacks [17]. Most embedded systems are implemented as System on a Chip devices, where all important system components (processor, memory, I/O) are implemented on a single chip. This solution extends the functionality of such systems to include both reconfigurable hardware and a continuous monitoring system that guarantees secure operations. Through monitoring, abnormal behavior of the system can be detected and hardware defense mechanisms can

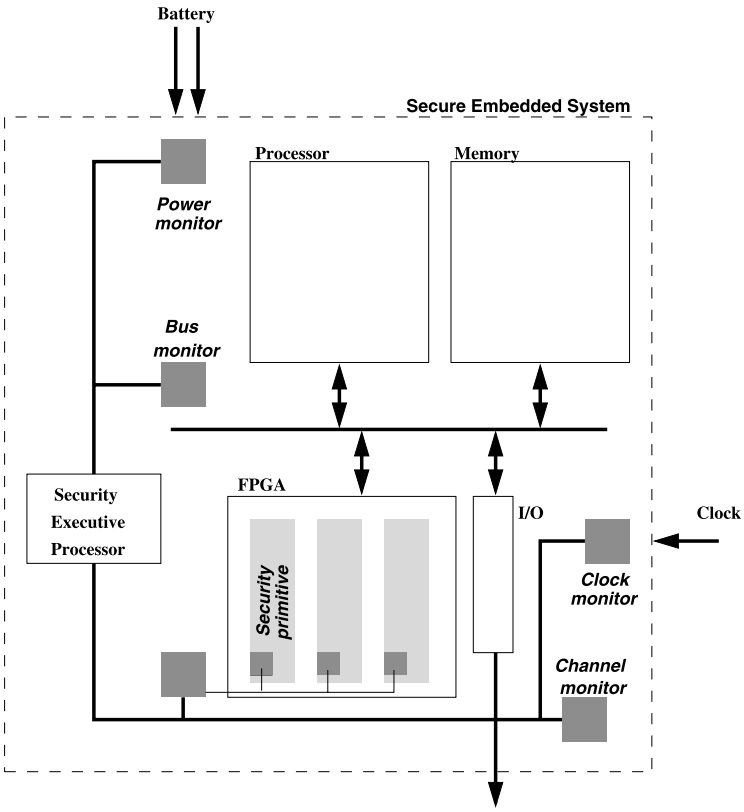

Fig. 2.10 The Security Architecture for Embedded Systems. The reconfigurable architecture contains the security primitives and the monitors protect the system

be used to fend off attacks. Such an approach has several advantages since application verification and protection is provided by dedicated hardware and not inside the application. The security mechanisms can be updated dynamically depending on the application running on the system, which guarantees the durability of the architecture. The SAFES approach focuses on embedded security and exploits the characteristics of embedded computations.

Figure 2.10 provides an overview of the architecture. As we can see, several monitors are used to track system-specific data. The number and complexity of the monitors are important parameters as they are directly related to the overhead cost of the security architecture. The role of these monitors is to detect attacks against the system. To provide such a solution, the normal activity (i.e. correct or expected) behavior of the modules is characterized in such a way that irregular behavior is detected. Autonomy and adaptability have been stressed to build an efficient security network of monitors. The monitors are autonomous in order to build fault tolerant systems; if one monitor is attacked, the others continue to manage the security of the

system. The monitors are distributed so as to be able to analyze the different parts of the system (e.g. battery, buses, security primitives, communication channels).

Different levels of reaction, reflex or global, are used depending on the type of attack. A reflex reaction is performed by a single monitor, in this case the response time is very short since no communication is required between the different monitors. A global reaction is performed when an attack involves a major modification of the system. In this case, the monitors need to define a new global configuration of the system, which requires a longer response time. The monitors are linked by an on-chip intelligence network. This network is controlled by the Security Executive Processor (SEP) that acts as a secure gateway to the outside world. The SEP provides a software layer to map new monitoring and verification algorithms to monitors. This point is important to meet FIPS 140-2 requirements, which require that data I/Os and security I/Os are implemented through separate links. In response to abnormal behavior, the SEP can issue commands to control the operation of the system. For example, it can override the power management or disable I/O operations.

Detecting an attack is a good thing, but a good response to a threat is of course needed. In order to qualify for FIPS 140-2 security level 2, the secret information has to be deleted in a short time. In an FPGA context this could be a problem if cipher keys are stored in the bitstream, because deleting bitstream memory could take too long for FIPS recommendations. For FIPS 140-2 levels 3 and 4, the task is even harder, as the entire secured device has to be destroyed. Such devices are rare and are reserved for government use, for example for nuclear weapons. A common reaction is to trigger explosives if an attack is detected, but chemical products can be used instead. These mechanisms could be used in the FPGA context, but another solution is possible. If the configuration memory and the FPGA content are erased, we can consider that the secured device is destroyed. Thus the job is dual-purpose, the FPGA content has to be destroyed, and the configuration memory has to be erased because it is the image of the underlying FPGA content. For SRAM FPGA devices, the first problem could be resolved by removing the FPGA power supply, which would result in the removal of the volatile configuration. In secured volatile FPGAs, erasing the bitstream memory can be avoided. In fact these devices embed a secret key used to decrypt the bitstream, and configuring data decryption is no longer possible when this key is erased. The need to erase configuration memory could be replaced by removing the bitstream secret key. As volatile FPGAs require a battery to maintain the configuration of the secret key, secured volatile FPGA devices could be considered destroyed if all power sources are removed (secure configuration battery and FPGA power supply). For non-volatile technologies, such a simple mechanism cannot be used because the secret key and bitstream data are not erased when the power is off. Of course configuration memory is not a problem for non-volatile FPGAs.

To perform a non-invasive attack such as DPA or DEMA, attackers need to collect a lot of leakage information computed with the same cipher key. The system could detect an abnormal use of a device by fixing a maximum number of operations allowed with the same key, for example. As previously mentioned, fault injection

could be detected at lower levels. But redundancy in attack detection is not a bad thing as it allows different types of attacks to be countered. With standard FPGAs, the user is limited by chip capabilities, most FPGAs do not have analog parts that can monitor environmental parameters. With the latest Actel FPGA, Fusion, various analog functions are available including temperature sensors, clock frequency or voltage monitoring. These functionalities can be used to detect possible fault injection.

In order to overcome standard FPGA deficiencies (no analog parts for example), one solution is to use tamper proof sealing, like IBM PCI Cryptographic Coprocessor product [21]. Invasive attacks are no longer possible, and the metal shield also protects against electromagnetic analysis attacks. With such a mechanism, many parts like analog sensors could be placed in a trusted environment near the FPGA, thereby extending the capabilities of the programmable device. This is not possible without tamper proof sealing because an attacker could freely act on analog sensors or even remove them.

One major concern with FPGAs at the system level concerns communications between the different resources within the system. Depending on the trusted area coverage within the system, it is essential to protect the communications and to minimize the overhead due to cryptography primitives. Some solutions are presented below. Their goal is to lessen the cost of confidentiality and integrity.

PE-ICE is a dedicated solution providing strong encryption and integrity checking to data transferred on the processor-memory bus of an embedded computing system [12]. It was designed to optimize latencies introduced by the hardware security mechanisms on read and write operations. To achieve this goal, PE-ICE uses a single block encryption algorithm. Data confidentiality is thus guaranteed by encryption while integrity checking relies on the spreading feature of block encryption algorithms and on resources available on chip.

Data privacy is provided by AES (Advanced Encryption Standard) encryption. AES is a block cipher algorithm that processes 128-bit blocks with a 128-bit key.

The spreading feature of block encryption algorithms implies that once a block encryption is performed, the position and the value of all bits in a ciphertext block C are influenced by each bit of the corresponding plaintext block P. Consequently if P is composed of two distinct data (PL and T), after ciphering, it is impossible to distinguish the PL ciphered part from the T part in C. Moreover if one bit is modified in C, after decryption all bits of the corresponding plaintext block will be affected.

PE-ICE integrity checking process: The previous property is used to add the integrity checking capability to block encryption. In a write operation (Fig. 2.11), a tag value is inserted in each plaintext block P before encryption. As a result, P is composed of a payload PL (data to protect) and a tag (T). Such a tag must be a nonce, a Number used ONCE, for a given encryption key and does not need to be calculated over the data with a specific algorithm; for example, it can be generated by a counter. After encryption, an indistinguishable and unique PL/T pair is created and the resulting ciphered block C is written in the external memory. In a read operation (Fig. 2.11) C is loaded and decrypted. The tag T derived from the resulting plaintext block is compared to an on chip regenerated tag called the reference tag T. If

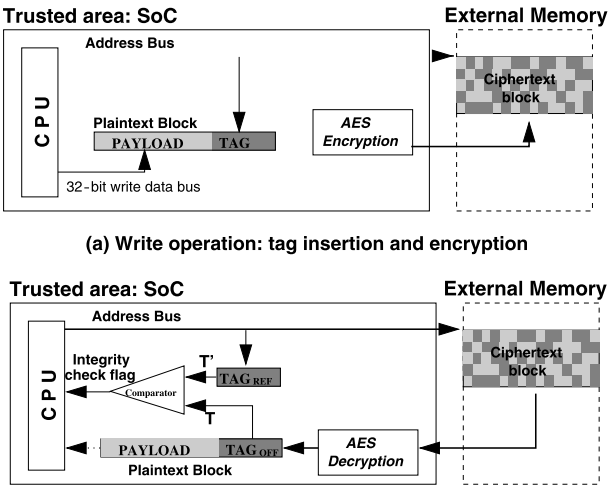

Fig. 2.11 PE-ICE principle: the protection is based on the spreading feature on block encryption algorithms

T does not match T, it means that at least one bit of C has been modified (spoofing attack), PE-ICE raises an integrity checking flag to prevent further processing.

The tag generation: In the context of processor—memory communication, the hardware engine executes both encryption and decryption. Therefore, the engine has to hold the tag value T of each ciphered block between encryption and the decryption or alternatively, must be able to regenerate it in read operations to perform the integrity checking process. The challenge is to reach this objective by storing as little tag information as possible on the engine to optimize on chip memory usage. Moreover T may be public knowledge because an adversary needs the secret encryption key to create an accepted PL/T pair. CPUs process two kinds of data, Read Only (RO) data and Read/Write data (RW). The composition of the tag differs for each kind of data and depends on their respective properties. RO data are only written once in external memory and are not modified during program execution. In addition, the secret encryption key is changed for each application downloaded in the external memory. Therefore, the tag can be fixed for each plaintext block of RO data. PE-ICE uses the most significant bits of the ciphered block address as tags (Fig. 2.11.a). If an attacker launches a splicing attack, the address used by the processor to fetch a block and by PE-ICE to generate the tag reference (T) will not match the one loaded as the tag (T). RW data are modified during software execution and are consequently sensitive to replay attacks. Using only the address as the tag is not enough to prevent such attacks because the processor cannot check if the data stored at a given address is the most recent data (temporal permutation). For that reason, the tag is composed of the most significant bits of the address of each ciphered block, concatenated with a random value (Fig. 2.11.b). The random value

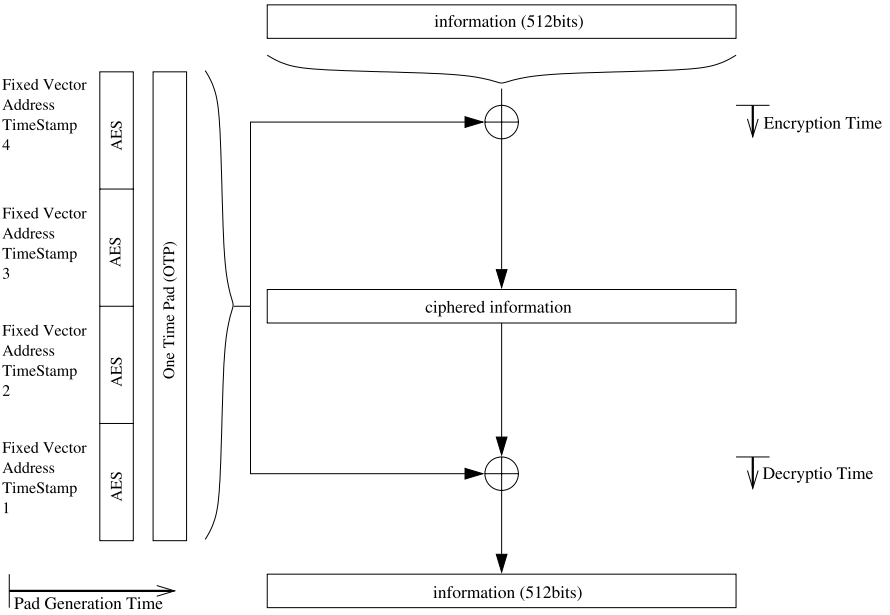

Fig. 2.12 One Time Pad model: ciphering and deciphering are performed through a XOR operation

of each plaintext block is changed for each write operation and is stored on chip to be able to regenerate the tag reference during read operations.

The OTP (One Time Pad) approach is an alternative solution to PE-ICE that tries to further parallelize the decryption time and the fetch latency. If we consider a block of 512 bits, i.e. the same size as the cache line that corresponds to a keystream and that is created using an AES algorithm, this block is created with the address of the data, a timestamp and a fixed vector (these data are required to protect the memory against classical attacks, spoofing, relocation, and timing). Except for the timestamp, all the information is known by the processor before memory access. Considering the possibility of using a special (fast) cache for timestamp storage, while the processor is fetching the data, a hardware engine computes the keystream (required for OTP). When the data are ready, the computation to decipher the information is only a bit-a-bit XOR gate (principle of OTP). Figure 2.12 shows the general model and the benefit of using OTP compared to classical ciphering techniques. If the memory access time is longer than the AES computation time, the deciphering latency is completed hidden by the model, as can be seen in Fig. 2.13.

Code compression techniques can also be used to reduce the cost of cryptography [48]. This solution aims to overcome the main problem of memory encryption i.e. the encryption delay. In general the decryption unit is defined in the secure area of the system, between the processor cache and the main memory, so that, by observing the bus activity, an attacker will be able to find encrypted information. This approach will impose an overhead in the processing time due to the deciphering unit.

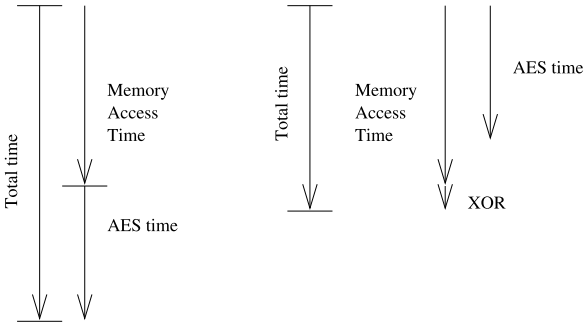

Fig. 2.13 OTP access time: keystream generation time is hidden by the memory access time. Ciphering is performed through a XOR operation

Using code compression, the number of transfers between the cache and the main memory will be reduced, which means that the global decryption overhead will be minimized, and if a scheme that naturally improves performance is used, the overhead can be completely avoided.

The general idea is presented in Fig. 2.14. After the compression phase, the AES algorithm is used to encrypt the blocks of 4×32 bits (the same size as a cache line). The compressed and encrypted code is then loaded into the main memory. When the processor asks for an instruction, the decompressor asks the cache if it is available. In this case, no deciphering is necessary. If the instruction in the cache is a codeword, the decompressor acts by delivering the corresponding instructions. If the instruction

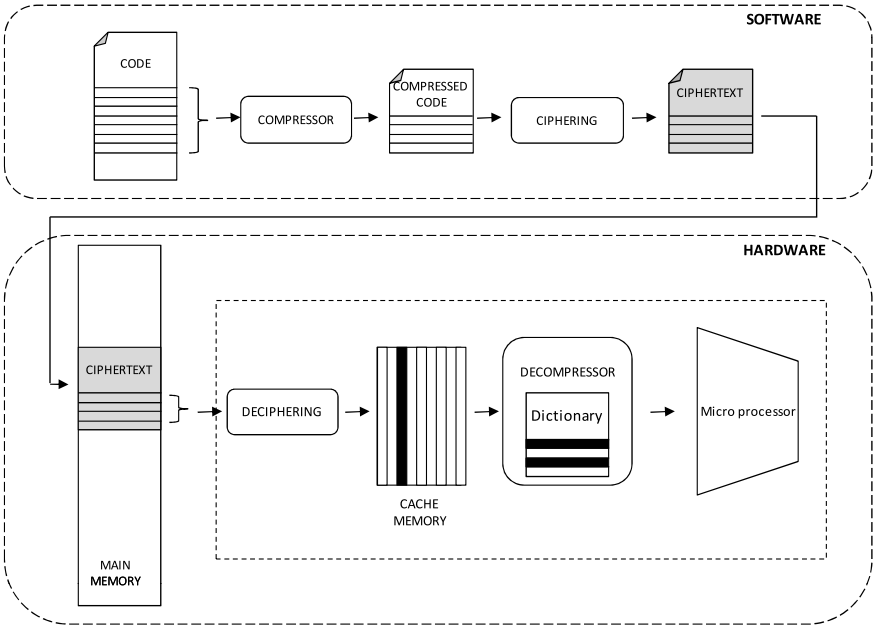

Fig. 2.14 Code compression techniques to mitigate the cost of cryptography

is not present in the cache, the deciphering unit is used, retrieving the block from the main memory and converting it into a compressed code that is delivered to the cache. This method relies on the efficiency of the compression to avoid some cache misses and, consequently accesses to the main memory. Moreover, the density of the code that is transferred through the bus is higher, so that less deciphering is required for the whole execution of the code.

All the solutions presented above allow for a more efficient and secure design at the system level. These solutions could also be considered for ASICs and are not fully dedicated to FPGAs. However, the advantages of FPGAs should not be considered as dealing only with a single level but with the whole security pyramid. In such a case, the different contributions e.g. dynamic reconfiguration at all the levels (i.e. technological, logic, architecture and system) and logic security improvements provide some very interesting features and should encourage the use of FPGAs for secure embedded systems.

2.7 Conclusions

Embedded systems are currently facing an increasing number of attacks since the amount of digital private information embedded in these systems is getting bigger every day. The rate of digital data exchanged is also increasing exponentially and transferred information must be kept secure since confidentiality and integrity are mandatory. Embedded systems are playing an essential role in our society and are already included in many electronic devices from low end to high end systems. Security is a serious problem and attacks against these systems are becoming more critical and sophisticated. New solutions are needed to allow the definition of secure embedded systems since current technologies are facing several challenges and cannot cope with security requirements and performance constraints. Architectures will have to meet high performance requirements, be energy efficient, flexible, tamper resistant and reliable to enable their wide adoption. FPGAs can address these requirements and provide efficient security primitives. Their characteristics enable the system to prevent attacks or to react when attacks are detected while guaranteeing the required energy and computation efficiency.

In this chapter we have analyzed current threats against embedded systems and especially FPGAs. We have described the standard FIPS requirements to build a secure system. This point is of paramount importance as it guarantees the level of security of a system. We have also highlighted current vulnerabilities of FPGAs at all the levels of the security pyramid. From a design point of view, it is essential to be aware of all these levels in order to provide a comprehensive solution. As mentioned earlier in this chapter, the strength of a system is defined by its weakest point; there is no reason to enhance other means of protection, if the weakest point is not tackled. Many serious attacks target this weakness in order to avoid facing the complexity of brute force attacks. Several solutions have been outlined in this chapter especially at the logical, architectural, and system levels to find a global solution.

References

1. Abraham, D.G., Dolan, G.M., Double, G.P., Stevens, J.V.: Transaction security system. IBM Syst. J. (1991)
2. ACTEL: Proasic3/e security. In: Application Note. ACTEL Corporation (2005)
3. Agrawal, D., Rohatgi, P., Rao, J.R.: Multi-channel Attacks. Lecture Notes in Computer Science (2003)
4. Bajard, J.-C., Imbert, L.: Leak resistant arithmetic. In: Cryptographic Hardware and Embedded Systems, Proceedings of CHES (2004)
5. Barker, E., Roginsky, A.: NIST special publication 800-131a—transitions: recommendation for transitioning the use of cryptographic algorithms and key lengths. In: Computer Security Division—Information Technology Laboratory (January 2011)
6. Barthe, L., Benoit, P., Torres, L.: Investigation of a masking countermeasure against side-channel attacks for RISC-based processor architectures. In: FPL, pp. 139–144 (2010)
7. Ciet, M., Nevel, M., Peeters, E., Quisquater, J.-J.: Parallel FPGA implementation of RSA with residue number systems—can side-channel threats be avoided. In: UCL Crypto Group (2004)
8. Coburn, J., Ravi, S., Raghunathan, A., Chakradhar, S.: SECA security-enhanced communication architecture. In: Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (2005)
9. Dagon, D., Martin, T., Starner, T.: Mobile phones as computing devices: the viruses are coming! IEEE Pervasive Comput. **3**(4) 11–15 (2004)
10. Dandalis, A., Prasanna, V.K.: An adaptive cryptographic engine for Internet protocol security architectures. ACM Trans. Des. Autom. Electron. Syst. **9**, 333–353 (2004)
11. Daniel, M., Techer, J.-D., Torres, L., Robert, M., Cathebras, G., Sassatelli, G., Moraes, F.: Current mask generation: an analogical circuit to thwart DPA attacks. In: IFIP International Federation for Information Processing (2006)
12. Elbaz, R., Champagne, D., Gebotys, C., Lee, R., Potlapally, N., Torres, L.: Hardware mechanisms for memory authentication: a survey of existing techniques and engines. In: Transactions on Computational Science IV. Lecture Notes in Computer Science, vol. 5430. Springer, Berlin (2009)
13. Elbirt, A.J., Yip, W., Chetwynd, B., Paar, C.: An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. IEEE Trans. Very Large Scale Integr. **9**, 545–557 (2001)
14. FPGA: Reverse engineering services. http://www.bltinc.com/services.fpga-reverse-engineering.htm
15. Gaj, K., Chodowiec, P.: Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays. In: Proc. RSA Security Conf. (2001)
16. Giraud, C.: DFA on AES. Technical report, Oberthur Card Systems (2004)
17. Gogniat, G., Wolf, T., Burleson, W., Diguet, J.-P., Bossuet, L., Vaslin, R.: Reconfigurable hardware for high-security/high-performance embedded systems: the safes perspective. IEEE Trans. Very Large Scale Integr. **16**(2) 144–155 (2008)
18. Gueron, Sh., Parzanchevsky, O., Zuk, O.: Masked inversion in GF($2n$) using mixed field representations and its efficient implementation for AES. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
19. Guilley, S., Pacalet, R.: SoC security: a war against side-channels. Annals of the Telecommunications. Système sur puce Électronique pour les Télécommunications (2004)
20. Hodjat, A., Verbauwhede, I.: A 21.54 Gbits/s fully pipelined AES processor on FPGA. In: Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines (2004)
21. IBM: IBM PCI cryptographic coprocessor. In: General Information Manual. IBM Corporation (2002)
22. Järvinen, K.U., Tommiska, M.T., Skyttä, J.O.: A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In: Proceedings of the ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (2003)

23. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2002)
24. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. Lecture Notes in Computer Science (1996)
25. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. Lecture Notes in Computer Science (1999)
26. Liardet, P.-Y., Teglia, Y.: Fault resistance: from reliability to safety. In: Proceedings of the International Conference on Dependable Systems and Networks (2004)
27. Martin, T., Hsiao, M., Ha, D., Krishnaswami, J.: Denial-of-service attacks on battery-powered mobile computers. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (2004)
28. Mcloone, M., Mccanny, J.V.: High performance single-chip FPGA Rijndael algorithm implementations. In: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (2001)
29. Mcmillan, S., Cameron, C.: JBITS implementation of the advanced encryption standard (Rijndael). In: Proceedings of the International Conference on Field-Programmable Logic and Its Applications (2001)
30. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: CHES, Worcester, MA, USA, August 17–18. LNCS, vol. 1965, pp. 71–77. Springer, Berlin (2000)
31. Nash, D., Martin, T., Ha, D., Hsiao, M.: Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops (2005)
32. Nassar, M., Bhasin, S., Danger, J.-L., Duc, G., Guilley, S.: BCDL: a high performance balanced DPL with global precharge and without early-evaluation. In: DATE'10, Dresden, Germany, March 8–12, 2010, pp. 849–854. IEEE Comput. Soc., Los Alamitos (2010)
33. National Institute of Standards Technology: Security requirements for cryptographic modules (FIPS pub 140-2). In: Federal Information Processing Standards Publication. National Institute of Standards and Technology (NIST) (2001)
34. Örs, S.B., Oswald, E., Preneel, B.: Power-analysis attacks on an FPGA—first experimental results. In: Proceedings of the CHES 2003 (2003)
35. Popp, T., Mangard, S.: Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In: Proceedings of CHES'05, Edinburgh, Scotland, UK, September 2005. Lecture Notes in Computer Science, vol. 3659, pp. 172–186. Springer, Berlin (2005)
36. Razafindraibe, A., Robert, M., Maurine, P.: Analysis and improvement of dual rail logic as a countermeasure against DPA. In: PATMOS, Göteborg, Sweden, pp. 340–351 (2007)
37. Saggese, G.P., Mazzeo, A., Mazzocca, N., Strollo, A.G.M.: An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In: Proceedings of the International Conference on Field-Programmable Logic and Its Applications (FPL 2003) (2003)
38. Schaumont, P., Verbauwhede, I.: Domain specific tools and methods for application in security processor design. Des. Autom. Embed. Syst. **7**, 365–383 (2002)
39. Standaert, F.-X., Örs, S.B., Preneel, B.: Power analysis of an FPGA: implementation of Rijndael: is pipelining a DPA countermeasure. In: Proceedings of the CHES 2004 (2004)
40. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D.: Efficient implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs. In: Proceedings of Cryptographic Hardware and Embedded Systems (2003)
41. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D.: A methodology to implement block ciphers in reconfigurable hardware and its application to fast and compact AES Rijndael. In: Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (FPGA 2003) (2003)
42. Standaert, F.-X., Örs, S.B., Quisquater, J.-J., Preneel, B.: Power analysis attacks against FPGA implementations of the DES In: Proceedings of the FPL 2004 (2004)
43. Tiri, K., Verbauwhede, I.: Synthesis of secure FPGA implementations. In: Proceedings of International Workshop on Logic and Synthesis (IWLS 2004) (2004)

44. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: Design Automation and Test in Europe Conference (DATE 2004) (2004)
45. Trichina, E.: Combinational logic design for AES subbyte transformation on masked data. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
46. Trichina, E., Korkishko, T.: Secure AES hardware module for resource constrained devices. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
47. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: CHES. Lecture Notes in Computer Science, vol. 3156, pp. 1–15. Springer, Berlin (2004)
48. Wanderley, E., Vaslin, R., Gogniat, G., Diguet, J.-P.: A code compression method to cope with security hardware overheads. In: 19th IEEE International Symposium on Computer Architecture and High Performance Computing (2007)
49. Wollinger, T., Paar, C.: How secure are FPGAs in cryptographic applications. In: Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (2003)
50. Wollinger, T., Paar, C.: Security aspects of FPGAs in cryptographic applications. In: Lysaght, P., Rosenstiel, W. (eds.) New Algorithms, Architectures and Applications for Reconfigurable Computing, pp. 265–278. Springer, Dordrecht (2005)