Cost-Effective Design Strategies for Securing Embedded Processors

Florent Bruguier, Pascal Benoit, Lionel Torres, Lyonel Barthe, Morgan Bourrée, Victor Lomné

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

Cost-Effective Design Strategies for Securing Embedded Processors

Florent Bruguier, *Member, IEEE,* Pascal Benoit, *Member, IEEE,* Lionel Torres, *Member, IEEE,* Lyonel Barthe, Morgan Bourree, Victor Lomne

Abstract

Side-Channel Attacks (SCAs) such as Differential Power or ElectroMagnetic Analysis (DPA/DEMA), pose a serious threat to the security of embedded systems. In the literature, few articles address the problem of securing General Purpose Processors (GPPs) with resourceful countermeasures. However, in many low-cost applications where security is not critical, cryptographic algorithms are typically implemented in software. Since it has been proved that GPPs are vulnerable to SCAs, it is desirable to develop efficient mechanisms to ensure a certain level of security. In this paper, we extend side-channel countermeasures to the Register Transfer Level (RTL) description. The challenge is to create a new class of processor that executes embedded software applications, which are intrinsically protected against SCAs. For that purpose, we first investigate how to integrate into the datapath two countermeasures based on masking and hiding approaches. Through an FPGA-based processor, we then evaluate the overhead and the effectiveness of the proposed solutions against time-domain first-order attacks. We finally show that a suitable combination of countermeasures significantly increases the side-channel resistance in a cost-effective way.

Index Terms

Cryptography, Side-Channel Attacks, RISC Processor, Countermeasures, Masking, Hiding, FPGA, Time-Domain

EDICS Category: 3-BBND

- *F. Bruguier, P. Benoit, L. Torres, L. Barthe, M. Bourree, and V. Lomne,are with the Microelectronics Department of Montpellier Laboratory of Informatics, Robotics and Microelectronics (LIRMM), UMR 5506, University of Montpellier - CNRS, Montpellier, France. E-mail: secnum@lirmm.fr*

Cost-Effective Design Strategies for Securing Embedded Processors

1 Introduction

1.1 Context

To meet the stringent security requirements of electronic systems, various cryptographic tools have been developed. Cryptography, in its traditional approach, studies the security of an algorithm as a mathematical function. For that purpose, it assumes a *black-box* model, in which attackers have only access to inputs and outputs of algorithms in order to obtain the secrets, the cryptographic keys. However, when these algorithms are executed on hardware devices that have the ability to store, process, and output digital data, adversaries have also the opportunity to tamper, perturbate, and spy their physical implementation.

As a consequence, new threats have been identified. In particular, attacks based on run-time information gained from physical characteristics of cryptographic implementations have received an ever increasing interest since their public introduction in 1996 by Paul. C. Kocher [1]. These cryptanalysis techniques, referred as *Side-Channel Attacks (SCAs)*, exploit different forms of information leakage such as the power consumption, electromagnetic emanations, or execution time from integrated circuits. Because such side-channels are dependent on the processed data and operations, simple and complex statistical analyses of collected samples can be conducted to retrieve the encryption keys of cryptographic algorithms.

With the discovery of SCAs, the security at the physical level has thus become a major priority for designers and developers of secure embedded systems. To prevent, or at least to mitigate the potential of these attacks, both industrial and research organisations have investigated a number of solutions, called *countermeasures* [2]. Although most of these methods do not guarantee absolute security, they can significantly reduce the side-channel leakages to make the attacks impractical [3]. However, regardless of the technique used, the implementation of countermeasures does not come for free. Most hardware and software solutions introduce considerable power, area, and performance overheads that are not suitable for resource-limited embedded systems.

1.2 Objective

Most of the papers in the literature are only focused on cryptoprocessors, *i.e.* dedicated hardware cryptographic cores for encryption and decryption processing. However, in various applications, cryptographic algorithms are directly implemented as pieces of software in General Purpose Processors (GPPs) for cost and flexibility reasons. Generally, for such applications, the required security is not the highest one, but it is important though to guarantee a certain level of trust. Within this context, the present paper investigates the security issues related to software cryptographic implementations running on embedded processors. More precisely, the attention is focused on the threat of time-domain first-order power and electromagnetic analysis attacks that predict intermediate values such as the *Differential Power Analysis (DPA)* [4] and *Correlation Power Analysis (CPA)* [5].

The objective is to enhance efficiently at low-cost the robustness of the core of embedded processor architectures by introducing design strategies at the Register Transfer Level (RTL), so that the executed software is intrinsically protected against these attacks. The RTL approach is motivated by the need to offer attractive solutions that are independent of the target technology. The real challenge is to provide efficient countermeasures that strike the balance among security, area cost, computing performance, and power consumption in order to meet the requirements of embedded systems. Another fundamental step is to describe the development of a prototype system based on Field-Programmable Gate Array (FPGA) technology, which implements the proposed methods. This approach has indeed the considerable advantage of allowing a security evaluation process under real-environment conditions, which is all the more relevant in the framework of SCAs.

1.3 Contributions

This paper presents novel design strategies to mitigate SCAs on embedded processors. Even if they are based on known principles, their application and their implementation on general processor architectures constitutes the originality of this work. The major contributions include:

- a *masking* countermeasure that conceals intermediate values with random values throughout the pipelined architecture of embedded processors (Section 2),
- a *hiding* countermeasure that efficiently randomises the execution of operations at the data-path level by exploiting micro-architectural functionalities of processors (Section 3), and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

- a practical evaluation of the Side-Channel Resistant (SCR) implementation of the *SecretBlaze* [6], an open-source 32-bit soft-core processor developed to investigate the problem of time-domain first-order SCAs on embedded architectures (Section 4).

2 Investigation of a Masking Countermeasure

The masking has been extensively studied by both academic and industrial research groups. Its principle is based on *secret sharing*, in which intermediate values are shared by means of random numbers called *masks*, such that each share alone is independent of the secret. For instance, an i^{th}-order masking scheme shares each intermediate value in $(i+1)$ shares with i random masks. Several masking schemes have been proposed in the literature, most targeting block ciphers [7], [8], [9], [10], [11]. Such countermeasures have become popular as their soundness can be formally proven [12].

In this context, we propose to study and extend the principle of masking to the datapath of embedded processors. For that purpose, we only consider the 1^{st}-order masking scheme, which is briefly introduced in the next subsection.

2.1 General Description

2.1.1 1^{st}-order Masking Principle

To remove the dependence between the side-channel leakages and the internal *sensitive values* of a cipher implementation, the fundamental idea of a 1^{st}-order masking scheme is to share a sensitive data d into two shares; a mask m and a *masked data* d_m such that:

$$d \rightarrow (d \circ m,\ m) = (d_m,\ m) \tag{1}$$

where $\circ$ is a *group operation*, which can be for instance Boolean or arithmetic.

From (1), the original value can be obtained from the inverse element according to:

$$(d_m,\ m) \rightarrow d = d_m \circ m^{-1} \tag{2}$$

For convenience, we use the terms *to mask* and *to unmask* to refer to equations (1) and (2), respectively.

When m is randomly and uniformly chosen from d, each of the shares is independent of d. As a consequence, the side-channel leakages of the overall cipher execution are independent of the secret, which thus provides a protection against 1^{st}-order statistical power and electromagnetic analyses.

The challenge of this approach lies in the difficulty to reconstruct the expected value: this step is the *mask correction*. Indeed, for a proper masking scheme, both the mask and the masked data have to be uniformly distributed throughout the various processes of a cryptoalgorithm implementation. There are two cases to consider:

- when the transformation process has a linear system property, and
- when the transformation process has a non-linear property.

For the first case, a linear transformation $\mathfrak{L}$ can be applied independently to each share according to equation (3):

$$(\mathfrak{L}(d \circ m), \mathfrak{L}(m)) = (\mathfrak{L}(d) \circ m', m') \tag{3}$$

where $m' = \mathfrak{L}(m)$.

Hence, the correct value can be extracted at the end of the process by computing its inverse transformation, as illustrated by equations (4) and (5):

$$\forall d, m,\ \ \mathfrak{L}(d \circ m) = \mathfrak{L}(d) \circ m' \tag{4}$$

$$\Rightarrow \mathfrak{L}(d) = \mathfrak{L}(d \circ m) \circ m'^{-1} \tag{5}$$

For the second case, the masking structure becomes more complex due to the non-linear property of the transformation. To overcome this problem, a common technique is to modify the non-linear part in order to produce the expected value. The secure solution is to pre-compute and store the values resulting from all possible combinations of masks and masked values into special *masked tables* [13]. Although the overhead of this approach is not negligible, it has the advantage to guarantee the correct execution of a cryptographic algorithm without involving the use of sensitive data.

To be an effective method, it is essential to remark that a particular attention should be given to operations with masked data and mask values. For instance, if two masked intermediate values are processed, we need to ensure that the result is still masked, *i.e.* it is crucial to avoid intermediate data sharing the same mask. In addition to this problem, all steps related to the mask correction should be carefully implemented to limit the information leakage. In practice, the usefulness of the masking method may be compromised if these details are not properly considered.

2.1.2 Dual Pipelined Datapath Masking Scheme

As the masking technique is an efficient method to remove the dependence between processed data and the side-channel leakages, we suggest to study the integration of its algorithmic description into the architecture of embedded processors at the RTL. Our solution consists of implementing a *dual pipelined datapath*.

Basically, the idea is to introduce a special datapath for the mask itself, which can be coupled to a classic RISC-based datapath. Hence, instead of directly handling raw data, the processor operates on a dual datapath with masked data. The main role of the new datapath is to keep the corresponding mask for each masked data along the pipeline structure of the processor. It thus allows to implement all steps related

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

to the mask correction to ensure the correct execution of instructions. Besides, one or more Pseudo Random Number Generators (PRNGs) are also included to generate the masks, which should be updated at each step of the datapath for a more efficient masking scheme.

The simplified model of our approach is depicted in Figure 1. Green (long) dash lines illustrate the pipeline with masked data, blue dot lines indicate the pipeline with masks, whereas black (short) dash lines point out the optional hardware. The direction of the data flow is indicated by arrowheads. In addition, filled circle arrows are used to denote the interaction between each datapath to properly implement the mask correction.

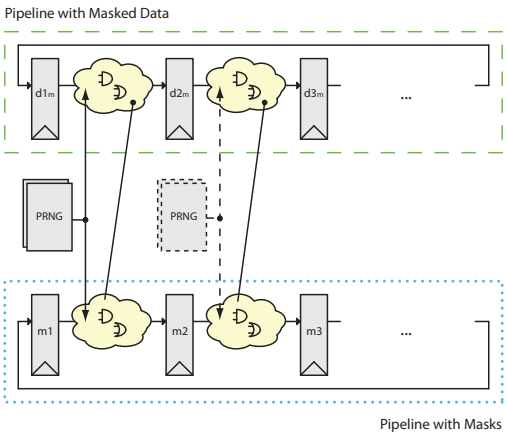


Fig. 1. The simplified model of the dual pipelined datapath masking scheme.

Despite the apparent simplicity of the concept, there are significant challenges to overcome, in particular, about the mask correction and the memory management of the masks.

2.2 Architectures

To implement the idea of the dual pipelined datapath, different approaches can be considered, along with their advantages and disadvantages. In the following paragraphs, we present various architectures of an embedded masked processor that attempts to address the side-channel issues. Thanks to a RISC-based design, we examine the masking strategy for two cases:

- the datapath related to register-to-register instructions, and
- the datapath related to memory-reference instructions.

2.2.1 Masked Datapath for Register-to-Register Instructions

The first datapath to consider is the one that takes place in the core of a processor: the computational datapath, which is exclusively used by register-to-register instructions. It starts from the register file, then goes through several pipeline stages, performs mathematical operations, and finally returns back to the register file.

With the proposed masking strategy, the processor implements a new datapath dedicated to masks, operating in parallel to the original datapath. It thus follows the same steps, starting from fetching operands from the register file and ending to store the result into the register file. A *register file of masks* is provided to store each mask associated with the respective masked data. The architecture also includes various *pipeline registers of masks*, carrying the mask value from one stage to another.

Implementing the mask correction is more challenging. For this purpose, we distinguish two categories of register-to-register instructions:

- arithmetic and logic instructions, and
- control flow instructions.

In case of arithmetic and logic instructions, masked data are transferred from stage to stage without any loss of data integrity until the execute phase, where all mathematical operations are implemented. While masked cryptographic co-processors usually involve to pre-compute and store masked tables for specific operations, the large number of instructions and their relative complexity require a trade-off between the cost in terms of gates and the efficiency of the masking method. As a consequence, we propose, in a first step, to perform ALU operations with unmasked data and, in a second step, to mask the result of the ALU with a new Pseudo Random Number (PRN). This crucial design choice is also motivated by the fact that, even if SCAs are still effective on combinational logic, experimental results suggest that the leakage at the register stages is predominant [14], [15] and masking ALU operations results in large overhead [16]. Hence, although this solution is not perfect, it has the advantage to protect most critical parts of the datapath while achieving an attractive trade-off between performance and security.

In case of control flow instructions, the mask correction is straightforward. Since they are not the target of the studied model of attacks, it is therefore allowed to unmask the masked data for all related processes (computations, address assignments, branch evaluations, etc.) without breaking the efficiency of the masking scheme.

Figure 2 illustrates the proposed masked datapath for register-to-register instructions. The direction of the data flow is indicated by arrowheads while the optional hardware used to update the mask is depicted with black dash lines. Unmask and mask modules are also included in the execute phase of the pipeline. Note that, if the execute phase requires more stages, the same approach should be adopted. Finally, filled circle arrows are used to denote other steps of the mask correction that are used for control flow instructions and do not provide useful information for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

side-channel analysis.

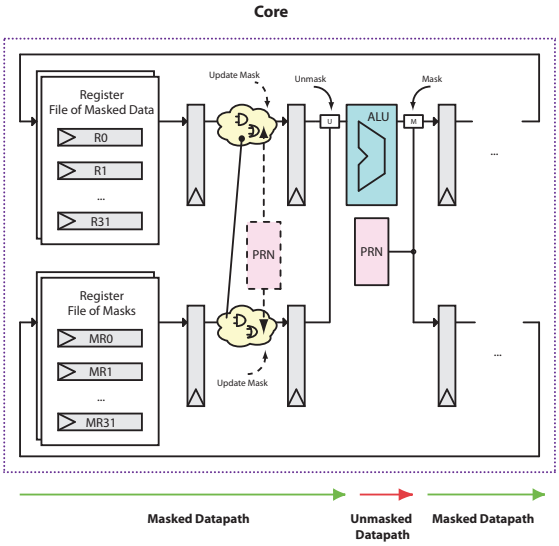


Fig. 2. The architecture of the masked datapath for register-to-register instructions.

2.2.2 Masked Datapath for Memory-Reference Instructions

The second datapath that leaks sensitive information comes from memory-reference instructions. As a load-store architecture, memory instructions transfer data between the register file and the memory sub-system of the processor.

Like the masked datapath for register-to-register instructions, the idea of the countermeasure is to secure the data flow of the architecture by introducing a new pipeline of masks with various pipeline registers throughout all steps of load and store instructions, starting from the register file and ending to the memory sub-system of the processor. To reduce the overhead of the solution, some parts of the masked datapath for such instructions can be shared with the one for register-to-register instructions. The structure of the data alignment process also facilitates the integration of the masking scheme into the core of the processor, reducing thus the complexity of the mask correction for load and store instructions.

In this article, the memory management of masks that depends on the depth and complexity of the memory hierarchy is not addressed in detail. A simple approach that consists in generalising the dual pipelined concept to the whole memory architecture is assumed. Hence, all registers and RAMs used by the data part of the memory sub-system have to be duplicated to store the masks associated to the masked data. This concerns not only the data cache memory, but also all components linked to the cache architecture such as the memory controller, the bus structure, and the main memory of the system. More efficient approaches are discussed in detail in [17].

TABLE 1
Performance and resource overhead of the masking countermeasure.

	Without Masking	With Masking	Overhead
# Flip-Flops	897	1234	+37.6%
# LUT	2423	3389	+39.9%
#BRAM	18	22	+22.2%
fMAX (MHz)	64.5	52.4	−18.8%

TABLE 2
Performance and resource overhead of the SecretBlaze's PRNG.

	SecretBlaze's PRNG
# State Bits	1024
# Random 0utput per cycle	32
Maximum SR length	32
# Flip-Flops	350
# LUT	510
fMAX in MHz	125

2.3 Implementation and Overhead Evaluation

Choosing the *SecretBlaze* [6] – a 32-bit embedded processor – as a case study, we implemented the concept of the masked datapath based on the Boolean group. At the hardware level, this group has not only a low overhead cost, but also the advantage to reduce the complexity of the integration of masking and unmasking operations into the pipelined architecture of the processor.

The performance impact and the resource overhead of the studied masking method were estimated in Table 1. A cost-effective configuration of the SecretBlaze (instruction and data caches were set to 8 KB and the core included a barrel shifter unit) was chosen to conduct this evaluation. The generation process of the mask values was implemented using a PRNG optimised for FPGAs that produces three 32-bit mask values each clock cycle. This PRNG is based on the use of Shift Registers (SRs). It takes advantage of bit-wise XOR operations and the ability to turn Look-Up Tables LUTs into SRs, providing a good balance between quality and area. The complete description of the PRNG can be found in [18].

These results were obtained with Xilinx's XST 12.1 using the 90 nm Spartan-3 technology (speedgrade -4). Synthesis options were set at the highest speed optimisation level with strong timing constraints. The resource usage was evaluated with the number of flip-flops, 4-inputs (LUTs), and Block RAMs (BRAMs) while the maximum operating frequency (fMAX) was estimated after the place and route process. Note that the results of the PRNG are given separately in Table 2. Clearly, a better (or worse) PRNG would lead to higher (or lower) resource requirements.

Not surprisingly, the overhead on the number of flip-flops is significant, owing to the introduction of several 32-bit pipeline registers for the mask values.

As regard the number of LUTs, the overhead is about 40% and mainly results from the introduction of the register file of masks (synthesised as distributed resources with 484 LUTs). The number of BRAMs also increases by 22% with 8KB dedicated to the cache of masks. Finally, the fMAX is reduced by 19%, which comes from special mask modules implemented within some critical paths.

In summary, the overhead costs introduced by the proposed masking method are relatively small. Furthermore, the SecretBlaze does not implement debug modules, exceptions and exception handling, a power management system, and a Memory Management Unit (MMU), which are not related to the masking countermeasure. That is why the performance impact and the resource overhead of the masking method for a similar industrial processor should be even smaller.

2.4 Comparison with Related Work

To the best of our knowledge, there is one example in the literature of a masking countermeasure that takes place into the datapath of embedded processors.

In [19], authors provide a complete masking framework for the LEON3 processor based on the Boolean masking with a secure zone (note that the original idea was introduced early in [20] with the LEON2 processor). However, their approach is only focused on the protection of cryptographic instruction set extensions and does not address the problem for the whole Instruction Set Architecture (ISA). Despite its efficiency, this method has some drawbacks including the development of specific instruction set extensions for an application, the customisation of the compiler, and the overhead due to the use of DPA-resistant logic styles. Unlike our method, their solution is not generic and circumvents the side-channel issues of embedded processors by treating only the inner blocks related to cryptographic extensions.

3 Investigation of a Hiding Countermeasure

The goal of hiding countermeasures is to make the physical characteristics of integrated circuits independent of intermediate values and operations performed during cryptographic applications. Among hiding countermeasures, we essentially distinguish two strategies: one based on the randomisation of the execution of cryptographic algorithms [21], [22] and one based on balanced DPA-logic styles [23], [24]. Note that, unlike masking-based countermeasures that are only efficient against statistical power and electromagnetic-based attacks, methods based on the randomisation of the execution increase the robustness of cryptosystems against most SCAs, including simple power analysis and timing-based attacks.

To be exhaustive in our study, we have chosen to study and integrate a hiding countermeasure that operates into the datapath of embedded processors.

3.1 General Description

Because of the large overhead of secure logic styles, our research efforts were focused on the integration of a cost-effective module that randomises the instruction stream at the hardware level.

Our idea is essentially based on the concept of a *non-deterministic processor* [25], in which the software can be executed with random additional operations that are generated by the hardware architecture of the processor. It allows not only to randomly insert dummy cycles that change the execution time, but also to randomise the usage of available hardware resources in order to increase the "noise", scrambling the patterns in the power consumption as well as the electromagnetic waves to prevent advanced signal processing, statistical, and modelling methods [26]. The countermeasure hence affects both time and amplitude dimensions of the physical leakages.

One solution consists of implementing a *pipeline randomiser* that handles a RISC datapath in a non-deterministic fashion through dummy control and data signals. Within this structure, a PRNG is required to provide a random information that should be used by the pipeline randomiser for taking a decision for each instruction being processed at each stage of the pipeline: either to keep a normal execution or to perform a dummy cycle. Additional PRNGs can also be implemented to generate random numbers for data operands during dummy cycles. This increases the random switching activity in order to lower the information leakage of the circuit.

The simplified model of our approach is depicted in Figure 3, in which the direction of the data flow is indicated by arrowheads and the additional hardware is illustrated with black (short) dash lines.

The challenge of this method lies in the difficulty to randomise both the execution of instructions and the content of registers in an efficient manner without altering the behaviour of the application.

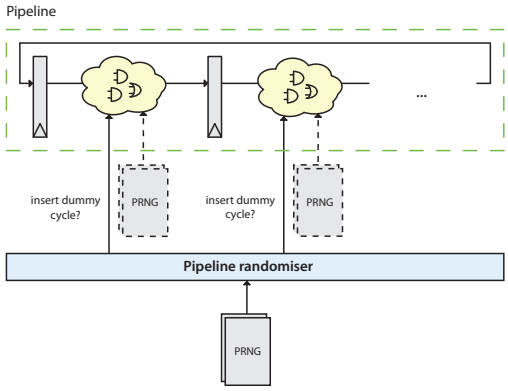


Fig. 3. The simplified model of the pipeline randomiser.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

3.2 Architecture

In this section, we introduce an efficient and cost-effective architecture to implement the pipeline randomiser method within the datapath of embedded processors. For that, we examine two key elements:

- the mechanism that implements a non-deterministic execution, and
- the mechanism that increases the noise.

3.2.1 Non-Deterministic Execution

When considering most RISC architectures of embedded processors such as ARM, MIPS, and PowerPC, a pipeline interlock is usually implemented to detect and solve data dependencies or control flow conditions specified in a sequence of instructions, known as *hazards*. This mechanism is often called a *hazard controller*, and thus manages the state of each pipeline stage throughout the whole architecture. A common technique to solve hazards is to *stall* all registers related to a pipeline stage until the data dependency is cleared (also known as pipeline bubbling). Sometimes it is also necessary to *flush* the pipeline stage to remove the execution of an instruction. Note that in case of an out-of-order execution (typically superscalar architectures for high-end embedded systems), advanced methods such as *scoreboarding* [27] or *Tomasulo algorithm* [28] are implemented to reduce the number of pipeline stalls. Nevertheless, a similar mechanism with stall, flush, and enable control signals remains necessary to ensure the correct execution of instructions.

To implement a non-deterministic execution, we can take advantage of the hardware features of the hazard controller (stall and flush control signals) to insert dummy cycles along the pipelined datapath. Indeed, although stall and flush are used primarily for hazards management, they can be also reused to insert dummy cycles using the same control logic, which does not allow to identify dummy cycles from control and data hazards. This design choice greatly reduces the overhead costs of the countermeasure (area, power, and may not affect the fMAX of the design) while achieving its purpose. Hence, by adding one or more false conditions to the controller, the pipeline operates in an unexpected way that can be efficiently exploited to insert additional calculations. We called it the *ghost hazard generation*.

To be an efficient technique, ghost hazards should be randomly generated. Nevertheless, an excessive use of random cycles can seriously decrease the throughput of the architecture. In practice, a suitable compromise needs to be found. To meet user application requirements, the probability of ghost hazards should be configurable by the software. One solution is to implement a module that compares a threshold value defined in a control register with a PRN generated at each clock cycle. The sign of the result hence determines the generation of ghost hazards. Note also, for real-time systems, it can be important to add a counter that determines the maximum number of consecutive dummy cycles. This can help to estimate the worst case execution time.

The architecture of the ghost hazard controller is illustrated in Figure 4. The direction of the data flow is indicated by arrowheads.

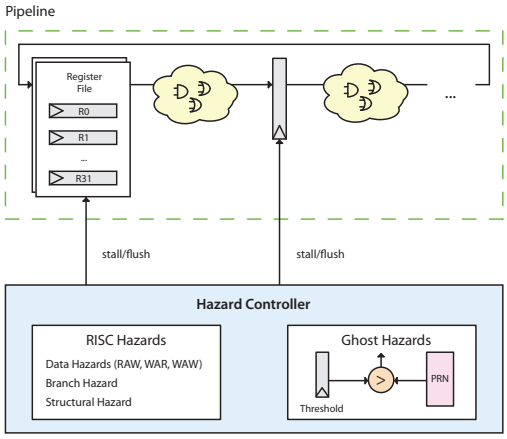


Fig. 4. The architecture of the ghost hazard controller.

3.2.2 Noise Generation

Non-deterministic processor can also introduce random additional calculations to lower the side-channel leakages. Like the masking countermeasure, we consider to randomise the architecture for two cases:

- the datapath related to register-to-register instructions, and
- the datapath related to memory-reference instructions.

The final architecture of the noise generation mechanism is depicted in Figure 5.

3.2.2.1 Randomised Datapath for Register-to-Register Instructions: For register-to-register instructions, a simple approach is to enable the propagation of random data operands from the register file to several functional units. It can be easily done by inserting additional muxes at the beginning of one or more pipeline stages. Hence, through the use of PRNs, pipeline buffers as well as functional units can be randomised to thwart an attacker. To ensure the integrity of the instruction flow, the designer has to ensure that the architecture does not write-back the result of a dummy calculation to the register file, for instance by forcing the write-back control signal to a no-operation state (by definition, this is the purpose of a flush signal).

For the sake of clarity, we would like to precise that most of functional units of the ALU (AND, OR, XOR, ROTATE, ADD, CMP, etc.) operate in parallel. This implies that, according to the type of the instruction being executed, only the result of the decoded

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

functional unit is stored into the pipeline register. In other words, all computations are done at the same time with more or less the same operands. It is hence not necessary to randomly modify all control signals during a dummy cycle (for instance, the one used to select the result from the functional unit). It also appears difficult to distinguish most logical and arithmetic instructions by simple analysis. Typical exceptions are multiply and divide instructions, which are usually handled as multi-cycle instructions with enable control signals to reduce the power consumption of these modules. Depending on the complexity of the processor architecture, it can also be more powerful to randomly activate such modules.

3.2.2.2 Randomised Datapath for Memory-Reference Instructions: Instructions that perform operations with the memory sub-system can seriously modify the leakages of a circuit by activating large memory banks. It is thus essential to add dummy memory operations for increasing the noise. A simple approach is to insert dummy load operations that does not write-back the result into the register file. For that, the address is randomly computed using a PRN and then is assigned to the data memory sub-system. Hence, depending on the available hardware, it may active the data cache memory sub-system, internal memories, or request a data from peripheral devices. In case of incomplete/partial memory map, the designer has to ensure that the random address is properly handled by the decoder of the processor and does not lead to a deadlock state.

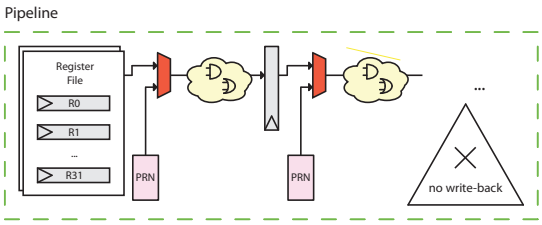


Fig. 5. The architecture of the noise generation mechanism.

3.3 Implementation and Overhead Evaluation

Both ghost hazard and noise generation mechanisms were implemented into the 5-stage pipelined architecture of the SecretBlaze. Unlike the masking countermeasure that requires attention to detail with mask values, the randomisation of the instruction stream can be done using a cost-effective design with a PRNG producing a 32-bit value each clock cycle without opening security holes. We arbitrarily chose to implement the noise generation mechanism during the execute phase of the SecretBlaze's pipeline, which significantly affects the side-channel leakages because of multiple functional units. Furthermore, the ghost hazard mechanism was implemented using the same source of PRNs to reduce the cost of the countermeasure.

The performance impact and the resource overhead of the hiding method were estimated in Tables 3 and 4. The evaluation was conducted using the same processor configuration and synthesis options to those described in Subsection 2.3.

TABLE 3
Performance and resource overhead of the hiding countermeasure.

	Without Hiding	With Hiding	Overhead
# Flip-Flops	897	910	+1.4%
# LUT	2423	2653	+9.5%
# BRAM	18	18	0.0%
fMAX in MHz	64.5	62.7	−2.9%

TABLE 4
Performance and resource overhead of the SecretBlaze's PRNG.

	SecretBlaze's PRNG
# State Bits	1024
# Random 0utput per cycle	32
Maximum SR length	32
# Flip-Flops	116
# LUTs	181
fMAX in MHz	125

According to these figures, it can be concluded that the proposed method has a negligible hardware overhead compared to conventional countermeasures. This observation mainly results from the ghost hazard mechanism that requires minor modifications to the original design. Synthesis reports also suggest that the increase in the number of LUTs is principally due to the introduction of larger muxes involved for the noise generation. At last, it should be noted that these tables do not include the execution time overhead, which is configurable by the software designer.

3.4 Comparison with Related Work

Non deterministic processors were introduced in [25] to thwart side-channel analysis. In this study, authors provide a framework to shuffle the instruction stream in a more or less random fashion. It takes advantage of an out-of-order execution mechanism, typically found in superscalar architectures. The proposed implementation was improved with a random register renaming technique described in [29]. The efficiency of the approach relies on the parallelism of the executed code that is, in practice, fairly limited. To address this issue, an additional pipeline stage was developed in [30], in which some random operations are generated without modifying the effective data. However, all these works can introduce a significant overhead and

can require both hardware and compiler tool-chain modifications.

An alternative method was presented in [31], where additional instructions are injected at run-time by the hardware. It requires small hardware and ISA modifications. Nevertheless, to reduce the complexity of the method, only a subset of instructions is allowed to be randomly executed, which clearly limits the non-determinism method as well as the noise generation.

Compared to these previous works, our contribution brings the following benefits. First of all, the proposed method aims to strike the balance between security and performance by exploiting available resources of most embedded processors such as the pipeline interlock mechanism. It also does not require the customisation of the software tool-chain with additional opcodes to generate dummy instructions. At last, it has the advantage to randomise the data-path at a low abstraction level (control and data signals rather than instruction codes), which gives the designer a greater control over the physical leakages.

4 Security Evaluation of the SecretBlaze-SCR Processor

In this section, we propose to evaluate the robustness of the masking and hiding countermeasures through prototyping. The motivation of this approach is to quantify the relative performance offered by the proposed masking and hiding strategies.

The lab instruments used in our experimental setup are a high-performance oscilloscope (3.5 GHz bandwidth, 40 GS/s sampling rate), a low-noise 63 dB amplifier, a near-field electromagnetic probe (500 μm diameter), a motorised X-Y-Z table allowing accurate positioning, and finally a Spartan-3 FPGA board. The whole measurement process and the data acquisition were controlled by a computer over RS232 and Ethernet communication protocols. All attacks were carried out on a desktop computer powered by a quad-core Intel Core 2 at 2.83 GHz with 8 GB of RAM. They were implemented through a MatLab program, in which the kernel was accelerated using C language through the use of MEX-functions. We detail in the following subsections the framework used for that purpose.

4.1 Processor Configurations

The SCR implementation of the SecretBlaze was synthesised at 50 MHz on a low-cost Spartan-3 FPGA. Several functional configurations of the SecretBlaze-SCR processor were defined to evaluate the benefit of the countermeasures separately, but also when combined. Note that it was crucial to ensure that the same FPGA bitstream (with fixed design and implementation characteristics) was used to allow a valid relative comparison between the different modes. Thanks to the flexibility provided by the countermeasures (control through enable bits), the SecretBlaze-SCR was evaluated for the following configurations:

- the *unprotected configuration*, used as the reference design,
- the *protected configuration with masking*,
- the *protected configuration with hiding*, and
- the *protected configuration with masking and hiding*.

Since our main objective was to achieve a secure implementation without jeopardising performance, the hiding countermeasure was arbitrarily configured to randomise the instruction flow with a maximum penalty of 35% (*i.e.* a weak randomisation in the context of side-channel analysis). More precisely, the probability to insert dummy operations was updated every 100 encryptions in software, varying the execution time overhead between 20% and 35%. The goal was to increase the efficiency of the proposed countermeasure by frequently changing the global distribution of dummy operations. Note also that the software overhead induced by the management of both masking and hiding strategies were negligible (less than 20 instructions to initialise and control the processor during the run-time execution).

4.2 Attack Scenario

The evaluation of the SecretBlaze-SCR was performed with the *Data Encryption Standard (DES)*, which has been the object of several studies regarding SCAs. The DES was implemented in C programming language, using a straightforward code that was not specifically optimised to take advantage of the 32-bit RISC architecture of the processor.

Using the Hamming Weight (HW) leakage model, the CEMA which is a time-domain attack was used as the reference model of attacks. This choice was mostly motivated by our previous experience with side-channel analysis on embedded processors, which has shown that the electromagnetic side-channel using the Pearson's correlation distinguisher provides the best results from an attacker's point of view.

4.3 Security Metrics

4.3.1 Measurement To Disclosure

Measurement To Disclosure (MTD) is the first metric that was introduced with the advent of SCAs. It is defined as the minimal number of power or electromagnetic traces required to correctly find the secret key. However, we recall that CEMA attacks rank among divide and conquer cryptanalysis methods that provide distinguishers for small key chunks, called subkeys, which can be recovered independently. In case of the DES algorithm, the attack divides the problem into 8 subkeys of 6 bits each. Consequently, there is a total of 64 combinations for each subkey, which gives a probability of 1.56% to pick

the correct one (assuming an equiprobable distribution). That is why, in practice, this metric is limited when applied for the 8 subkeys of the DES, since the probability to randomly find the correct subkey is clearly high. Nevertheless, MTD still can give an information about the efficiency of the attack, and thus was selected for the security evaluation process.

4.3.2 Measurement To Disclosure with Stability

To overcome the previous limitation, an extended version of the MTD metric was adopted: this is the *Measurement To Disclosure with Stability (MTDwS)*. MTDwS defines the amount of traces needed to guess the correct key with a stability criterion, which states that the distinguisher of an attack continuously output the correct key hypothesis. For that purpose, a threshold value is arbitrarily chosen. Due to the presence of countermeasures, we chose a value of 1000 traces to ensure high stability of the correct key hypothesis. The latter metric is particularly useful to estimate the statistical convergence of attacks.

4.3.3 Percentage of Correct Guesses

One restriction of the MTDwS metric is that the use of the stability as a criterion is not directly related to the security aspects. In the context of SCAs, the key recovery is mainly considered to evaluate the robustness of an integrated circuit. A classical approach is to measure the frequency of the correct key candidate according to the result of the distinguisher: this is the goal of the *Percentage of Correct Guesses (PCG)* metric. For more information about the effectiveness of the attack, we can also compute the percentage of guesses for other key hypotheses in order to determine the *rank* of the PCG.

4.3.4 Guessing Entropy

Even if the previously proposed metrics give us a good starting point to evaluate the countermeasures, these metrics are data-dependent. This might involve different results with a same set of measurements used in a different order. That is the reason why, we employ the *Guessing Entropy (GE)* metric to evaluate the different configurations of our processor [32]. This metric helps us to properly quantify weaknesses and to conclude whether or not the cryptosystem is successfully broken.

4.4 Experimental Results

Experimental results for each configuration of the processor are given in Table 5 while the CEMA traces obtained for the first subkey are depicted in Figure 6. We acquired 50,000 traces for the unprotected configuration, 100,000 traces for the protected configuration with masking, 100,000 traces for the protected configuration with hiding, and 200,000 traces for the protected configuration with masking and hiding. These acquisitions were done 3 times with 3 independent data sets. Note that, these data sets were divided into smaller data sets to calculate GE for the unprotected configuration and for the protected configuration with masking. During these experiments the bandwidth of the oscilloscope was adjusted at 2 GHz while the sampling rate was set at 40 GS/s. It took more than a week with our measurement setup to acquire all traces. The compressed traces required about 50 GB of disk space. It should be noted that we favored the quality of the measurements over the quantity.

Note that, the whole design was constrained using PlanAhead tool. We had therefore comprehensive knowledge of the spatial location of logic and sequential elements that implement the processor within the FPGA architecture. The probe was firstly located above the position of the pipeline registers. This one was finally adjusted after doing some measurements and computations before launching the whole measurement campaigns.

4.5 Analysis and Interpretation

4.5.1 Unprotected Configuration

The CEMA attack performed with 50,000 electromagnetic traces was successful. All subkeys were quickly broken, as evidenced by the three metrics (MTDwS, PCG, and GE): only 1,449 traces were necessary to fulfill the stability criterion for all subkeys while the PCG values reached more than 99%. Furthermore, the CEMA traces obtained for the subkey S1 (Figure 6(a)) highlight the security issues of the unprotected implementation by giving a time dimension to the leakage. The "accordion effect" clearly visible from the picture (black curve) demonstrates the vulnerabilities of the SecretBlaze-SCR's pipelined architecture.

4.5.2 Protected Configuration with Masking

From the figures given in the table, we first conclude that the SecretBlaze-SCR with the masking countermeasure offers a better resistance against CEMA attacks. The data associated to the stability criterion support this analysis. The MTDwS metric was fulfilled for all subkeys with 14,836 electromagnetic traces. Compared to the unprotected implementation of the processor, the MTDwS was thus increased by a maximum factor of 10.2 for this attack scenario. However, as evidenced by the results of average GE, all subkeys were recovered, which means that the CEMA attack was successful with less than 500 electromagnetic traces (Figure 7).

Despite a moderate improvement, the proposed masking countermeasure does not seem to provide a sufficient level of security for most applications. To figure out the reasons behind this observation, we propose to investigate the origin of the leakage by analysing the CEMA traces. From them (Figure 6(b)),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

TABLE 5
CEMA results.

Subkey #	S1	S2	S3	S4	S5	S6	S7	S8
MTD	141	101	101	144	101	165	108	219
MTDwS	1141	1104	1168	1243	1101	1389	1164	1449
PCG	99.75%	99.80%	99.72%	99.57%	99.80%	99.39%	99.74%	99.21%
Rank	1	1	1	1	1	1	1	1
Broken	**success**	**success**	**success**	**success**	**success**	**success**	**success**	**success**

(a) Unprotected configuration.

Subkey #	S1	S2	S3	S4	S5	S6	S7	S8
MTD	512	2392	801	4932	5325	101	101	10368
MTDwS	3057	4751	3045	14836	13784	5612	4293	11511
PCG	98.21%	96.93%	98.40%	88.98%	88.56%	95.67%	96.69%	81.49%
Rank	1	1	1	1	1	1	1	1
Broken	**success**	**success**	**success**	**success**	**success**	**success**	**success**	**success**

(b) Protected configuration with masking.

Subkey #	S1	S2	S3	S4	S5	S6	S7	S8
MTD	147	861	8111	8612	373	failure	32483	2394
MTDwS	24747	28733	14017	11376	59384	failure	83165	failure
PCG	74.33%	58.91%	74.76%	20.87%	39.63%	0.00%	15.66%	0.17%
Rank	1	1	1	1	1	63-64	1	62
Broken	**success**	**success**	**success**	**success**	**success**	failure	**success**	failure

(c) Protected configuration with hiding.

Subkey #	S1	S2	S3	S4	S5	S6	S7	S8
MTD	failure	23738	27465	failure	35008	111708	377	35856
MTDwS	failure	failure	failure	failure	156132	103069	failure	38305
PCG	0.00%	0.35%	0.23%	0.00%	7.87%	0.24%	0.09%	2.62%
Rank	64	58	62	63-64	3	59	61	8
Broken	failure	failure	failure	failure	failure	failure	failure	failure

(d) Protected configuration with masking and hiding.

it is apparent that the CEMA attack was successful on the subkey S1, but the leakage appears to be confined to a small number of operations. We can indeed observe two peak areas related to the correct subkey hypothesis. However, the "accordion effect" due to the pipelined architecture is no longer observable. By cross-referencing the behaviour of the SecretBlaze-SCR's pipeline with the list of the executed instructions, we were able to determine which part of the processor was still leaking sensitive information: the ALU was identified as the main source of the leakage. This observation is in fact not surprising. Indeed, it is the consequence of the design choices made during the integration of the masking scheme into the datapath of the SecretBlaze-SCR. We recall that, to achieve a cost-effective implementation, the processor performs ALU operations with unmasked data, which corroborates the results of the CEMA traces. Furthermore, these results confirm that all pipeline registers as well as the data memory sub-system are no longer leaking sensitive information thanks to the masking protection.

To conclude, the experiment evaluation demonstrates that our masking countermeasure is still vulnerable against first-order CEMA attacks. However, its most striking benefit is the confinement of the leakage to a small part of the processor, which modestly enhances the overall robustness of the processor architecture.

4.5.3 Protected Configuration with Hiding

From Table 6c, we can see that six subkeys were broken, but the ranks obtained for other two subkeys were very low. Besides, the subkey S6 was never found according to the MTD metric. To recover the full key, we should have made more measurements.

We thus conclude that the proposed hiding countermeasure works as expected, since the randomisation does not prevent SCAs, but makes them more difficult to perform. Clearly, the more dummy operations are inserted, the more the side-channel resistance of the processor is increased. This experimental evaluation also reveals the statistical effect of the proposed hiding countermeasure over the leakage. Indeed, by analysing the CEMA traces from Figure 6(c), we still are able to distinguish the "accordion effect" related to the pipelined architecture, reflecting a large number of critical operations handled by the processor. This observation suggests that the instruction stream is not sufficiently randomised to thwart an attacker, since

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2407832, IEEE Transactions on Emerging Topics in Computing

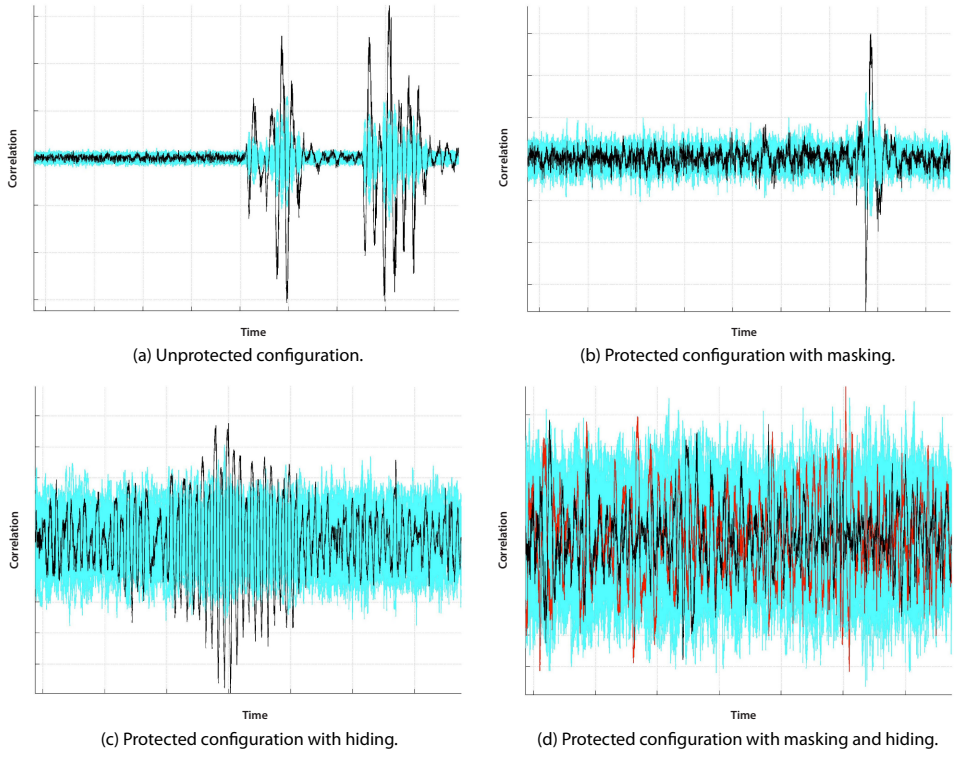


Fig. 6. CEMA traces obtained for the first subkey (*cyan* = wrong subkey hypothesis, *black* = correct subkey hypothesis, *red* = wrong guessed subkey hypothesis).

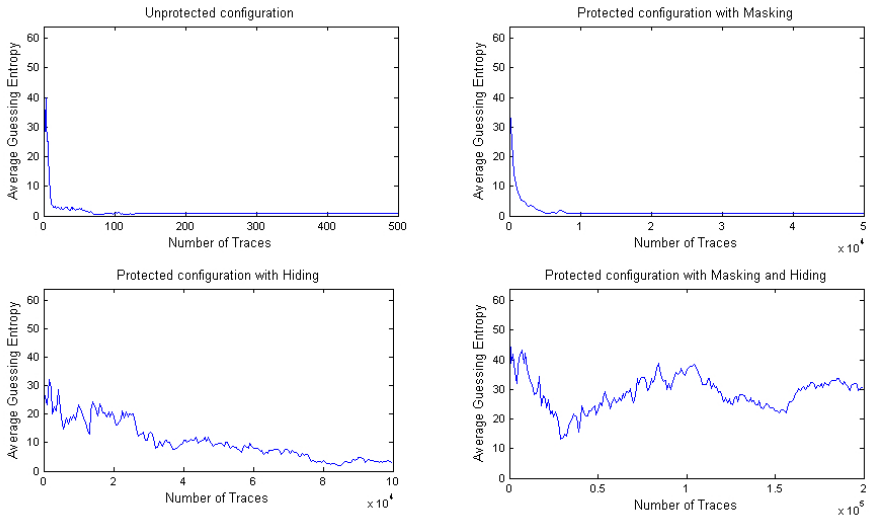


Fig. 7. Number of traces vs average guessing entropy.

the overlapping of critical instructions in the pipeline may intersect with a low randomisation.

4.5.4 Protected Configuration with Masking and Hiding

Using both masking and hiding countermeasures, the CEMA attack was not successful. This observation

proves that the robustness of the processor was deeply increased.

From an attacker's point of view, the best result was obtained for the subkey S5, which was correctly guessed many times, as indicated by the PCG metric (7.87%). It is difficult to conclude if this subkey was correctly found due to a residual leakage or due to a random situation. The margin, defined as the percentage of the difference between the amplitude of the highest correlation obtained for the guessed subkey hypothesis and the amplitude of the correlation obtained for other subkey hypotheses, was very low (less than 5%) for this subkey, which highlights again the efficiency of the countermeasures. Besides, the CEMA traces obtained for the first subkey support this analysis. From Figure 6(d), it is apparent that the correlation is not relevant, and compared to the previous configurations, the accordion effect is no more perceptible. These results are confirmed by the Figure 7 which shows that the average GE is not converging to one.

Hence, we conclude that the combination of masking and hiding countermeasures can offer a significant protection against CEMA attacks. This is all the more remarkable given that the randomisation of the execution was very low (as a reminder, the timing overhead was 35% at maximum).

5 Summary and Conclusions

The research works presented in this paper have been conducted to address the threat of power and electromagnetic SCAs that predict intermediate values. They have been specifically focused on software cryptographic implementations running on embedded GPPs.

Several strategies for securing embedded processors at the RTL have been examined. We have first introduced the concept of the dual pipelined datapath masking scheme, which allows to conceal intermediate values of cryptographic algorithms within embedded processor architectures. Then, we have suggested the concept of the ghost hazard generation, a cost-effective hiding countermeasure that randomises the flow of instructions. Both solutions have been explored to efficiently balance the security needs with the performance and resource overhead. An experimental processor implementing the proposed countermeasures, the SecretBlaze-SCR, has been developed on Xilinx's Spartan-3 FPGA technology. As summarised in Table 6, the performance evaluation of the resulting design has shown that the average overhead induced by the countermeasures is suitable for many embedded systems.

Then, we have performed a practical security evaluation of the SecretBlaze-SCR. From the results obtained with CEMA attacks, we have shown that each countermeasure moderately enhances the robustness against some statistical attacks. While the masking countermeasure has the advantage to confine the leaking operations to the ALU of the processor, the hiding strategy configured with a low randomisation (35% at maximum) also provides a protection by affecting both time and amplitude dimensions of the physical leakages. Furthermore, we have demonstrated the complementary of these methods that can be combined to significantly increase the side-channel resistance of the processor. For an attack scenario of a DES software implementation, we have experimentally shown that the security of the SecretBlaze-SCR was deeply increased.

TABLE 6
Performance and resource overhead of the countermeasures.

	Masked Datapath	Ghost Hazard Mechanism	Both
# Flip-Flops	+37.6%	+1.4%	+39.0%
# LUT	+39.9%	+9.5%	+48.9%
# BRAM	+22.2%	0.0%	+22.2%
fMAX in MHz	−18.8%	−2.9%	−18.9%

As a main conclusion, these research works have given some valuable and practical information about the design and implementation processes of a secure embedded processor. We have developed resourceful countermeasures against time-domain first-order SCAs. This contribution may be further investigated in future works, against frequency domain attacks as well as high-order analysis. Although the proposed countermeasures are based on well-known concepts, they have been developed at the RTL description of processor architectures in order to prove their feasibility on silicon while addressing the requirements of many embedded systems. They have the advantage of being independent of the executed cryptographic algorithms, which gives the designer an attractive degree of flexibility when designing a secure system. Due to their construction, they also offer full compatibility with most existing software and technological countermeasures to achieve a higher level of security.

References

[1] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology*, 1996, pp. 104–113.

[2] P. Kocher, "Design and Validation Strategies for Obtaining Assurance in Countermeasures to Power Analysis and Related Attacks," in *Proceedings of the NIST Physical Security Workshop*, 2005.

[3] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," in *Cryptographic Hardware and Embedded SystemsCHES 2000*. Springer, 2000, pp. 252–263.

[4] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology*, 1999, pp. 388–397.

[5] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Proceedings of the 2004 Cryptographic Hardware and Embedded Systems Workshop*, 2004, pp. 16–29.

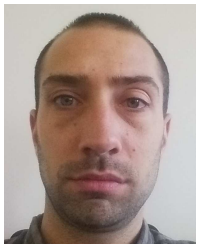
[6] L. Barthe, L. V. Cargnini, P. Benoit, and L. Torres, "A Configurable and Cost-Effective Open-Source Soft-Core Processor," in *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2011, pp. 305–308.
[7] L. Goubin and J. Patarin, "DES and Differential Power Analysis – The Duplication Method," in *Proceedings of the 1999 Cryptographic Hardware and Embedded Systems Workshop*, 1999, pp. 158–172.
[8] Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks," in *Advances in Cryptology*, 2003, pp. 463–481.
[9] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A Side-Channel Analysis Resistant Description of the AES S-Box," in *Fast Software Encryption*, 2005, pp. 413–423.
[10] K. Schramm and C. Paar, "Higher Order Masking of the AES," in *Topics in Cryptology, The Cryptographers Track at the RSA Conference*, 2006, pp. 208–225.
[11] E. Prouff and M. Rivain, "Provable Secure Higher-Order Masking of AES," in *Proceedings of the 2010 Cryptographic Hardware and Embedded Systems Workshop*, 2010.
[12] S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," in *Advances in Cryptology*, 1999, pp. 398–412.
[13] T. Messerges, "Securing the AES Finalists against Power Analysis Attacks," in *Proceedings of the 7th International Workshop on Fast Software Encryption*, 2000, pp. 150–164.
[14] B. Vaquie, S. Tiran, and P. Maurine, "A Secure D Flip-Flop against Side Channel Attacks," in *Proceedings of the 21st International Conference on Integrated Circuit and System Design: Power And Timing Modeling, Optimization, and Simulation*, 2011, pp. 331–340.
[15] H. Maghrebi, J.-L. Danger, F. Flament, S. Guilley, and L. Sauvage, "Evaluation of Countermeasure Implementations Based on Boolean Masking to Thwart Side-Channel Attacks," in *Proceedings of the 2009 International Conference on Signals, Circuits and Systems*, 2009.
[16] S. Tillich, M. Kirschbaum, and A. Szekely, "Implementation and evaluation of an sca-resistant embedded processor," in *Smart Card Research and Advanced Applications*. Springer, 2011, pp. 151–165.
[17] L. Barthe, "Stratgies pour scuriser les processeurs embarqus contre les attaques par canaux auxiliaires," Ph.D. dissertation, 2012.
[18] D. B. Thomas and W. Luk, "FPGA-Optimised Uniform Random Number Generators Using LUTs and Shift Registers," in *Proceedings of the 2010 Field Programmable Logic and Applications*, 2010, pp. 77–82.
[19] S. Tillich and M. Kirschbaum and A. Szekely, "Sca-resistant embedded processors: The next generation," in *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010, pp. 211–220.
[20] S. Tillich and J. Groschädl, "Power Analysis Resistant AES Implementation with Instruction Set Extensions," in *Proceedings of the 2007 Cryptographic Hardware and Embedded Systems Workshop*, 2007, pp. 303–319.
[21] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.
[22] R. Soares, N. Calazans, V. Lommé, A. Debahoui, P. Maurine, and L. Torres, "A Gals Pipeline Architecture to Increase Robustness against DPA and DEMA Attacks," in *Proceedings of the 23rd Symposium on Integrated Circuits and Systems Design*, 2010.
[23] K. Tiri, D. Hwang, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "Prototype IC with WDDL and Differential Routing - DPA Resistance Assessment," in *Proceedings of the 2005 Cryptographic Hardware and Embedded Systems Workshop*, 2005, pp. 354–365.
[24] V. Lomné, P. Maurine, L. Torres, M. Robert, R. Soares, and N. Calazans, "Evaluation on FPGA of Triple Rail Logic Robustness against DPA and DEMA," in *Design, Automation and Test in Europe Conference*, 2009, pp. 634–639.
[25] D. May, H. L. Muller, and N. P. Smart, "Non-Deterministic Processors," in *Australasian Conference on Information Security and Privacy*, 2001.
[26] F. Durvaux, M. Renauld, F.-X. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Cryptanalysis of the CHES 2009/2010 Random Delay Countermeasure," in *IACR Cryptology ePrint Archive*, 2012.
[27] J. E. Thorton, "Parallel Operation in the CDC 6600," in *AFIPS Proc. FJCC*, 1964, pp. 33–40.
[28] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development*, pp. 25–33, 1967.
[29] D. May, H. L. Muller, and N. P. Smart, "Random Register Renaming to Foil DPA," in *Proceedings of the 2001 Cryptographic Hardware and Embedded Systems Workshop*, 2001, pp. 28–38.
[30] J. Irwin, D. Page, and N. P. Smart, "Instruction Stream Mutation for Non-Deterministic Processors," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, 2002, pp. 286–295.
[31] J. A. Ambrose, R. Ragel, and G. Parameswaran, "RIJID: Random Code Injection to Mask Power Analysis based Side Channel Attacks," in *Proceedings of the 2007 Design Automation Conference*, 2007, pp. 489–492.
[32] F.-X. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," *Advances in Cryptology-Eurocrypt 2009*, pp. 443–461, 2009.

Florent Bruguier received a M.S. and PhD degrees in Microelectronics from the University of Montpellier, France, in 2009 and 2012, respectively. Since 2012, he is scientific assistant at LIRMM. He has co-autored over 20 publications. His research interests are focused on self-adaptive and secured approaches for embedded systems.

Pascal Benoit received a M.S. and PhD degrees in Microelectronics from the University of Montpellier, France, in 2001 and 2004, respectively. Then he joined the Karlsruhe Institute of Technology at the University of Karlsruhe in Germany where he worked as a scientific assistant. Since 2005, he is a permanent Associate Professor at LIRMM / University of Montpellier. He has co-authored over 130 publications in books, journals and conference proceedings, and holds 5 patents. His present research interests are self-adaptive and secured approaches for embedded systems.

Lionel Torres obtained respectively his Master and PhD degree in 1993 and 1996 from the University of Montpellier. From 1996 to 1997, he was in ATMEL company as IP core methodology R&D engineer. From 1997 to 2004, he was assistant professor at Polytech Montpellier engineering school and LIRMM laboratory. Since 2004, he is full Professor and was at the head of the Microelectronic department of the LIRMM from 2007 to 2010. He is now deputy head of Polytech'Montpellier in charge of research, industrial, and international relationship. His research interests and skills concern system level architecture, with a specific focus in the security and cryptographic applications and non-volatile computing based on emerging technologies. He leads several European, national and industrial projects in this field and is (co)author of more than 30 journal papers, 150 conference publications, and 7 patents.

Lyonel Barthe obtained his engineering degree in electronics at Polytech'Montpellier, France, in 2009, and his Ph.D in electrical and computer engineering at the University of Montpellier 2, Montpellier, France, in 2012. He is now working at Thales Alenia Space, Toulouse, France, as a FPGA designer for SATCOM modem solutions.



Victor Lomne is currently expert in hardware security at the hardware security lab of the ANSSI (french cybersecurity agency). He received the MS degree in cryptology and computer security from the University of Bordeaux, France, in 2007, and the PhD degree in electrical engineering from the University of Montpellier, France, in 2010. His research interests include embedded systems security, cryptographic implementations, physical cryptanalysis and hardware security.

Morgan Bourree obtained his engineering degree in electronics at Polytech'Montpellier, France, in 2008. From 2009 to 2011, he worked at IBM Burlington and at LIRMM until 2012. Since 2014, he is automotive consultant at Altran.