Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration

Nele Mentens[1,2], Benedikt Gierlichs[1], and Ingrid Verbauwhede[1]

[1] Katholieke Universiteit Leuven, ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{firstname.lastname}@esat.kuleuven.be
[2] Katholieke Hogeschool Limburg, IWT
Agoralaan Gebouw B bus 3, B-3590 Diepenbeek, Belgium
nele.mentens@iwt.khlim.be

Abstract. Dynamically reconfigurable systems are known to have many advantages such as area and power reduction. The drawbacks of these systems are the reconfiguration delay and the overhead needed to provide reconfigurability. We show that dynamic reconfiguration can also improve the resistance of cryptographic systems against physical attacks. First, we demonstrate how dynamic reconfiguration can realize a range of countermeasures which are standard for software implementations and that were practically not portable to hardware so far. Second, we introduce a new class of countermeasure that, to the best of our knowledge, has not been considered so far. This type of countermeasure provides increased resistance, in particular against fault attacks, by randomly changing the physical location of functional blocks on the chip area at run-time. Third, we show how fault detection can be provided on certain devices with negligible area-overhead. The partial bitstreams can be read back from the reconfigurable areas and compared to a reference version at run-time and inside the device. For each countermeasure, we propose a prototype architecture and evaluate the cost and security level it provides. All proposed countermeasures do not change the device's input-output behavior, thus they are transparent to upper-level protocols. Moreover, they can be implemented jointly and complemented by other countermeasures on algorithm-, circuit-, and gate-level.

1 Introduction

After the production of the first Complex Programmable Logic Devices (CPLD) and Field Programmable Gate Arrays (FPGA) in the 1980s, research in programmable devices has evolved in many directions. To put our idea in context, the following advances are worth mentioning. Partial reconfiguration increases the performance of a reconfigurable system by reducing the reconfiguration time. This can be done dynamically at run-time and without user interaction, while the static part of the chip is not interrupted. The idea we put into practice is a coarse-grained partially dynamically reconfigurable implementation of a cryptosystem. Our prototype implementation consists of a FPGA which is partially

E. Oswald and P. Rohatgi (Eds.): CHES 2008, LNCS 5154, pp. 346–362, 2008.
© International Association for Cryptologic Research 2008

reconfigured at run-time to provide countermeasures against physical attacks. The static part is only configured upon system reset.

Some advantages of dynamic reconfiguration for cryptosystems have been explored before. In such systems, the main goal of dynamic reconfigurability is to use the available hardware resources in an optimal way. This is the first work that considers to use a coarse-grained partially dynamically reconfigurable architecture in cryptosystems to prevent physical attacks by introducing temporal and/or spatial jitter. Note that the proposed countermeasures do not represent an all embracing security solution and should be complemented by other countermeasures.

The first experimental results of power analysis attacks on FPGAs were given by Örs *et al.* [20]. Standaert *et al.* examined the effect of pipelining and unrolling techniques on the power consumption of FPGAs [23]. Power analysis countermeasures based on the random pre-loading of pipelining registers are evaluated in [22]. Successful fault injection on FPGAs is reported by Maingot *et al.* in [16]. The concept of spatial jitter for hardware implementations has been addressed in [2] and [8], where architectures are proposed that consist of several identical elementary cells. An algorithm's suboperations are randomly mapped on these cells. In our solution, the suboperations are always performed in the same functional blocks, but these blocks are randomly relocated.

This paper is organized as follows: Section 2 gives an overview of the physical attacks and countermeasures relevant for this work. Section 3 describes the initial assumptions and the setup. Sections 4, 5, and 6 introduce the countermeasures temporal jitter, combination of spatial and temporal jitter, and fault detection for partially dynamically reconfigurable systems. Finally, Sect. 7 concludes the paper.

2 Physical Attacks and Countermeasures

Differential Side Channel Attacks (DSCA), as introduced by Kocher *et al.* [14], are passive attacks. They exploit the fact that there exists a relation between the bit-flips in an electronic cryptographic device and its instantaneous power dissipation. Since the bit-flips in the device depend on the values it is processing, and since these data depend on a secret, *e.g.* a cryptographic key, there exists a link between the secret and the power dissipation. First, an adversary observes the target device's power dissipation during the encryption of several messages X. She targets an intermediate result $f_{k_c}(X)$ of the cryptographic computation that depends on the known and varying data X and a (small) part of the secret key k_c. At the time instant t_c when this particular value is computed, there exists a significant correlation between the intermediate values $f_{k_c}(X)$ and the observed power dissipation $O(t_c)$. Since, in general, both t_c and k_c are unknown, the adversary performs an exhaustive search over all time instants t and key hypotheses k. For this search, she computes the values of $f_{k'}(X)$ based on a key guess k' and applies a power consumption model to derive hypothetical power consumption values $h(\cdot)$. Then, she applies a statistical test to measure

the correlation between the predicted and the observed power dissipation at all instants t. For one combination of the parameters t and k', the correlation will be significantly higher than for all others. This reveals not only the correct key k_c but also the time instant t_c when the targeted intermediate result is computed. We apply Correlation Power Analysis [7], predict the hypothetical power dissipation $h(f_{k'}(X), R)$ as the Hamming distance between $f_{k'}(X)$ and a reference state R, and use the Pearson correlation coefficient $\rho(h(\cdot), O(t))$ as statistical test.

In practice, countermeasures aim at making an attack more difficult and ideally infeasible. In this context infeasible means to raise the cost of an attack beyond the gain due to a success. A metric for measuring the difficulty of an attack is the number of samples required. Although this is not an ideal metric (the number depends on too many factors which are difficult to rate) it is often applied in practice. Hence, DSCA countermeasures aim at increasing the number of required samples. There exist many approaches to achieve this goal. They can be categorized along the implementation axis (algorithm-, circuit-, and gate-level) or according to their functionality (masking and hiding), see Table 1.

Table 1. Overview of Differential Side Channel Analysis Countermeasures

	Algorithm	Circuit	Gate
Masking	Algorithmic masking	–	Gate level masking
Hiding	Random precharge Dummy cycles Random Order Execution	Noise Generators Decoupled power supply	Dual-Rail Precharge Logic Current Mode Logic

Reference [18] presents a coarse-grained architecture that uses reconfigurability to provide an algorithmic masking scheme. In this work we focus on power analysis countermeasures that aim at introducing temporal jitter into the sequence of operations, *i.e.* distributing the instant t_c over time for several observations. Such countermeasures are effective because the intermediate result $f_{k_c}(X)$ is no longer computed at a fixed instance. It rather occurs at a set of different time instants $\mathcal{T}$ with probability distribution $\mathcal{P}$. Examples of this type of countermeasure for software implementations are Random Process Interrupts (Dummy Cycles) [10], and Random Order Execution [24]. Reference [1] presents a "Smart Processor" that inserts random delays autonomously and code independent. For hardware implementations the only countermeasure of this category we are aware of are asynchronous circuits [6]. Note that they can introduce vulnerabilities to timing attacks, since the execution time of the implementation might depend on the processed data itself. Moreover, asynchronous circuits generally require a longer design time than synchronous circuits. While the software countermeasures are easy to implement and virtually platform independent, an asynchronous circuit needs to be designed from scratch and implemented carefully.

Fault attacks are active attacks. In the broadest sense, they expose the target device to physical stress in order to provoke abnormal behavior. An additional information flow can be caused, if the cryptographic device returns erroneous cryptograms or a modified execution path is entered. The exploitation of faulty cryptograms may involve mathematical cryptanalysis. We distinguish between transient and permanent faults. A fault is transient if the device remains fully functional and the effect is of short duration (*e.g.* one clock cycle). A fault is permanent if its effect persists during the lifetime of the device. We also distinguish two classes of attacks. One class is composed of attacks that require a single successful fault injection to achieve the goal as for example the Bellcore attack [5] against a RSA-CRT implementation. Attacks in the other class usually require many successfully injected faults to achieve their goal. As examples we mention Collision Fault Analysis (CFA) [12], Differential Fault Analysis (DFA) [3], and Ineffective Fault Analysis (IFA) [9, 4].

Fault analysis countermeasures can be divided in at least three categories. Countermeasures of the first kind do not aim at preventing fault injection and the fault's effect, but intend to make the exploitation of the fault difficult and ideally infeasible. These countermeasures aim, as in the context of DSCA, at distributing the instant t_c at which a given operation is executed over a time interval. The second kind of countermeasure aims at detecting a fault injection by, for instance, introducing redundancy and checking for errors. This can be done at the data level using a suitable code and at the software level by executing the algorithm twice and comparing the results. In hardware, one can also implement the circuit twice and run both executions in parallel, or implement the circuit in dual-rail logic with a dedicated error state. The third kind of countermeasure aims at detecting the fault injection attempt. Usually, dedicated sensors are integrated into the circuit and/or the chip package.

In this paper we introduce countermeasures of all aforementioned types for partially dynamically reconfigurable devices.

3 Setup and Assumptions

3.1 Adversarial Model

The adversary is a malicious user of the device under attack, though she can be the legitimate owner. She wants to extract confidential data, *e.g.* cryptographic keys. The adversary can perform all kinds of passive attacks, in particular power analysis.

With respect to fault analysis, we apply the notions of the adversarial model introduced by Lemke-Rust and Paar in [15]. The adversary can also perform a range of active attacks, namely those categorized as semi-invasive. Summarizing this means, that the adversary can penetrate the device as much as to open the chip's package. Penetration of what is called the cryptographic boundary is not included. However, the adversary may use fault injection mechanisms that cross this line, *e.g.* photons.

An attack is successful, if enough key information is obtained to recover the entire key with or without further cryptanalysis. The adversary is able to inject at most q faults per second, where q is a small number. The adversary can use r fault injection setups in parallel where again r is a small number. Fault injection is a probabilistic process with success rate p. Complementary events with probability $1-p$ have no or not the intented effect and cannot be exploited. In specific attack scenarios, unsuccessful fault injection can even turn an attack infeasible because the adversary can not distinguish a cryptogram where the fault effect is as desired from a cryptogram where the fault effect is different.

Faults can be injected by precisely timed modifications of the clock signal or the power supply (glitches), intense illumination with focused white light or a laser beam, intense electromagnetic fields, rapid changes of the temperature, *etc.* Note that we exclude precise and deterministic permanent modification of the chip, *e.g.* cutting and re-wiring using a focused ion beam, from the adversary's capabilities. That is, we do not consider invasive adversaries. For all fault injection techniques, we assume that a successfully injected fault has a random and non-predictable effect on the targeted volume in the device. Although this makes the adversary appear weaker than she might be, it allows a compact analysis of the security level of the proposed countermeasures. Further, it is without doubt the most general and realistic model. Considering more specific fault models, such as deterministic bit-set and bit-reset, is beyond the scope of this work. To evaluate the adversary's success probability, we use the following definitions from [15].

Spatial resolution: let dA denote the target area at depth z with depth dz so that $dA \cdot dz$ is the target volume. ΔA is the area and Δz is the depth affected by the fault. The probability to stimulate the correct area on the chip surface is given as $p_{area} = 1$ if $\Delta A \leq dA$, and as $p_{area} = dA/\Delta A$ else. Since the penetration depth depends on various technological factors and cannot be estimated for a general case, we conservatively assume that the fault injection process always penetrates to the right depth, *i.e.* $p_{depth} = 1$. Combining the probabilities for area and depth therefore leads to $p_{volume} = p_{area}$.

Temporal resolution: let dt denote the targeted time interval during which a fault must be injected in order to be successful. Let ΔT denote the time resolution of the fault injection process. The probability to inject a fault at an instant where it leads to success is given as $p_{time} = 1$, if $\Delta T \leq dt$, and as $p_{time} = dt/\Delta T$ else.

The overall success probability is a function of (at least) these two probabilities. Thus in our case and given that they are independent $p = p_{volume} \cdot p_{time}$.

3.2 Reference Architecture and System Overview

To explain our countermeasures, we assume that the cryptographic algorithm to be implemented is a repetitive instruction that consists of a number of subfunctions. This is a realistic assumption for both symmetric and public key cryptosystems. Figure 1 shows the general architecture (bottom right) and floorplan (top right), respectively, consisting of n blocks each representing a subfunction.

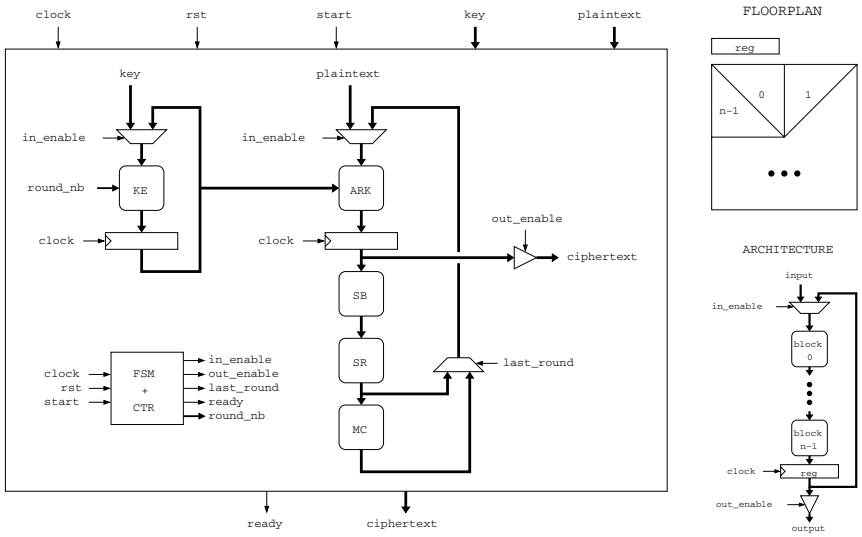

Fig. 1. General architecture and floorplan for the implementation of a repetitive algorithm consisting of n subfunctions (right) and reference architecture for our AES-128 prototype implementation (left)

These n blocks are executed a number of times and the intermediate result is saved in a register.

Moreover, we propose prototype implementations of AES with a 128-bit key [19]. The architecture of the fully parallel reference design of AES is depicted in Fig. 1 (left), where ARK, SB, SR and MC denote the subfunctions AddRoundKey, SubstituteBytes, ShiftRows and MixColumns, respectively. AES-128 consists of 10 rounds, where the round results are stored in an intermediate register. The roundkeys are computed on-the-fly using the KeyExpansion (KE) function and stored in the roundkey register. The intermediate register, the roundkey register, the multiplexors and the output buffer are controlled by the Finite State Machine (FSM) in combination with the round counter (CTR).

Our reference and prototype architectures are implemented on a Virtex-II Pro FPGA of Xilinx. In order to provide self-reconfiguration, an Internal Configuration Access Port (ICAP) [25] is added to the design. Figure 2 shows how a softcore MicroBlaze (μB) processor is connected to the partially reconfigurable AES coprocessor, the True Random Number Generator (TRNG) and the ICAP through the On-chip Peripheral Bus (OPB). The connection of the processor to the data and instruction memory (block RAM or BRAM) is realized over the Local Memory Bus (LMB). Since this paper only focuses on the security of the AES coprocessor, the bitstreams for our prototype implementations are stored in an external flash memory. More secure solutions include storing the bitstreams in the internal block RAM, although this has a limited capacity, or using secure external flash memory, as for example described by Handschuh and Trichina in [11].

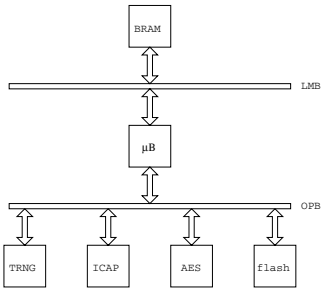

Fig. 2. Architectural view of the reconfigurable system

4 Temporal Jitter

As stated above, many attacks against physical implementations of cryptographic algorithms require that the timing of the executed operations is aligned for multiple executions. Since software countermeasures are flexible to apply and provide a high level of protection at the same time, we dedicate this section to the application of these well studied techniques to hardware implementations.

4.1 Description of a Generic Architecture

To port the idea of temporal jitter to hardware implementations, many registers could be foreseen in combination with multiplexors deciding whether to bypass a register or not. Because this would create a large overhead in resources, this option is highly impractical. We propose an architecture with a dynamically reconfigurable switch matrix to avoid such a problem. The matrix determines the position of one or more registers in between functional blocks. Since a register causes a delay of one clock cycle, randomly positioning registers in between subfunctions de-synchronizes the observations. Our architecture is shown in Fig. 3. It is an improvement of the reference architecture shown in Fig. 1 in Sect. 3.2.

The number of possible configurations depends on the number m of registers and the number n of blocks. The value n depends on the number of reasonable subfunctions in the algorithm, which may depend on the width of the data-path. The number of options increases if we allow cascaded registers in between functional blocks. This is shown in the third option for the switch matrix in Fig. 3. Note, however, that if we allow cascaded registers, there exist several configuration options that lead to identical sequences of combinatorial and sequential logic. Concerning this matter and allowing up to m cascaded registers, the number c of *distinct* configurations is $\binom{n+m-1}{m}$, *i.e.* the number of combinations of m elements out of n, where the order does not matter and repetition is allowed. The probability to observe the same configuration twice is $1/c$. However, the number of possible configurations determines the size of the memory needed to store the configuration data and is therefore bounded. Further, an increasing number of intermediate registers increases the number of cycles needed for one

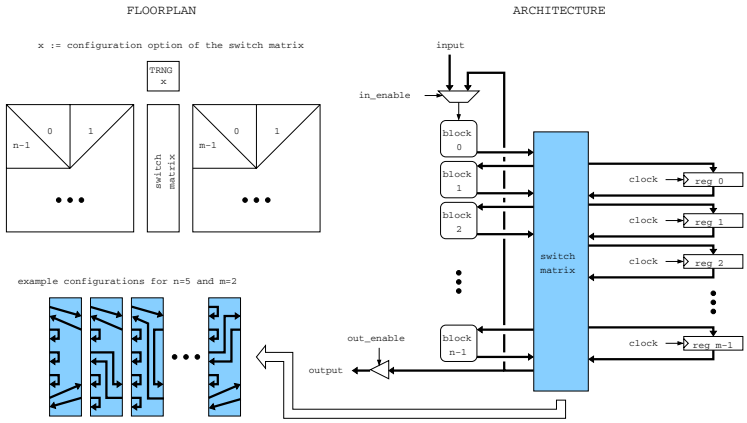

Fig. 3. Modified architecture for improved side channel analysis attack resistance

encryption. The number of registers, however, does not affect the maximal clock frequency, because we allow more than one register to be cascaded. In general, the number of options for the temporal shift is determined by the number m of registers and the number n of blocks, and is bounded above by c.

As illustrated in Fig. 3, a True Random Number Generator (TRNG) is used to select the next configuration of the switch matrix. It is important to note that the security of the architecture depends on strength of the TRNG and its resistance against fault and power analysis attacks. In this paper, we assume that the TRNG provides strong random numbers and withstands all adversaries covered by our model.

4.2 Example for AES-128 Encryption

In the fully parallel implementation of AES-128 in Fig. 1, four obvious subfunctions can be distinguished: AddRoundKey (ARK), SubstituteBytes (SB), ShiftRows (SR), and MixColumns (MC). We implemented a prototype based on the generic architecture proposed in the previous section where $n = 4$ and $m = 2$. The prototype and some options for the reconfiguration matrix are shown in Fig. 4. In this particular case and if we allow up to m cascaded registers in between functional blocks, the number of distinct configurations is $c = \binom{4+2-1}{2} = 10$.

The performance results of our implementation are compared to a static design in Table 2. The static design contains one register after each AES round, while the partially reconfigurable design contains $m = 2$ registers. The reconfiguration time of the switch matrix is approximately 3 ms. However, technological improvements reduce this number by a factor of at least 10. We also observed a decrease of the maximal clock frequency by a factor of more than 3. This is due to the communication between the static and the dynamic part of the design. The static part of the prototype design is larger than the fully static design. This

Table 2. Implementation results for the static design (one register) and the prototype dynamic design (two registers) on a Virtex-II Pro FPGA

	occupied area (# slices)	max. clock frequency (MHz)	through-put (Mbit/s)	reconf. time (ms)	reconf. data size (kB)	# conf. options
Static design	685(5%)	111	10			1
Prototype: static/dynamic	3251(23%) 1547(11%)/1704(12%)	33	1.5	3	91	10

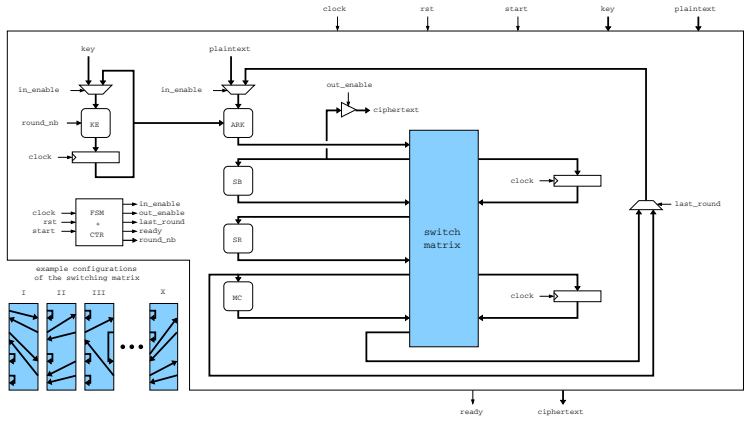

Fig. 4. Modified AES architecture for improved side channel analysis attack resistance

is because of the extra 128-bit register and because of the logic that is needed for the communication over the boundaries between the static and the dynamic part of the design.

4.3 Can the Countermeasure Be Circumvented?

An obvious approach to circumvent the countermeasure is to distinguish the different active configurations. If that is possible, an adversary can entirely undo the effect of the countermeasure by using only an appropriate subset of the observations that represent a single configuration. Therefore we examine, whether such a distinction is feasible using Timing Analysis (TA) [13] and Simple Power Analysis (SPA) [14].

The overall encryption time is constant and does therefore not reveal information about the circuit's internal configuration. The execution time is equal to $11 \cdot m$ cycles, where $m \geq 1$ is the number of intermediate registers.

Figure 5 shows power traces obtained from the prototype implementation while performing AES encryption in 2 out of 10 possible configurations. The two intermediate registers are pre-loaded with random data before the encryption starts. In this way, an attacker cannot deduce the position of the registers

from the height of the first two peaks. This technique is similar to randomly pre-loading registers in a pipeline, as described in [22]. The plots support our claim that all implementations execute the algorithm in the same constant time. Additionally one can see that both plots look very similar, though not exactly the same. The slight differences are due to the pre-loading with random data and they are not configuration dependent features. These observations hold for all 10 possible configurations. Hence the power traces do not allow the distinction of the circuit's internal configuration. However, both TA and SPA allow an adversary to find out the number m of registers, which was fixed at design time.

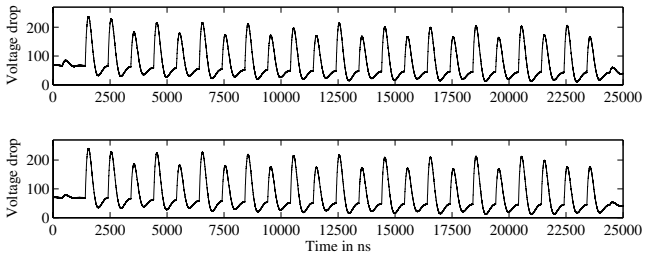

Fig. 5. Power traces of AES encryption in 2 configurations, with $n = 4$, $m = 2$ and a clock frequency of 1 MHz

Another approach to circumvent the countermeasure could consist of analyzing the circuit during dynamic reconfiguration. If an adversary can distinguish the different configurations, an attack as outlined above is feasible. We examine whether TA or SPA of the reconfiguration process might leak information about the circuit's current or next configuration. The spatial size of the reconfigurable area is constant and so is the size of the different bitstreams. Therefore, the timing of the entire reconfiguration process is constant and TA cannot reveal the circuit's current or next configuration. Similarly, the power consumption during reconfiguration does not show obvious configuration dependent features.

Without having exhausted all possibilities of TA and SPA, we assume that more elaborate analysis of both, the reconfiguration process and the actual encryption, does not allow to distinguish the configurations with a significant rate of success.

4.4 Resistance against DSCA

In this section we evaluate the level of protection that our countermeasure provides against DSCA. Recall that (standard) DSCA requires intermediate results to be synchronized in the time domain and that our countermeasure introduces time jitter.

In [17], Mangard studies the effectiveness of temporal de-synchronization as a DSCA countermeasure. He derives that, in the case of insertion of random delays, t_c is binomially distributed over time. Our proposal has the same effect.

Note that he implicitly assumes that t_c occurs at each $t \in \mathcal{T}$ equally likely, which is the most obvious choice. He derives a formula which allows to easily estimate the maximum correlation coefficient ρ_{max} as seen by an adversary. Taking his further simplifying steps into account and adapting to our notation, the equation becomes

$$\rho_{max} = \frac{\rho(h(f_{k_c}(X), R), O(t_c))}{\sqrt{1 + \frac{1}{SNR}}} \cdot \hat{p} = \rho' \cdot \hat{p} \tag{1}$$

where $\rho(h(f_{k_c}(X), R), O(t_c))$ is the correlation coefficient achieved for the correct key hypothesis k_c at the correct time instant t_c at an unprotected implementation and SNR is the signal to noise ratio. $\hat{p}$ is the the maximum probability in $\mathcal{P}$ and indicates the time instant $\hat{t}_c \in \mathcal{T}$ at which the targeted intermediate result occurs most likely. It is further possible to estimate the number S of samples required to break the protected implementation based on ρ_{max} and a quantile Z_α:

$$S = 3 + 8\left(\frac{Z_\alpha}{\ln(\frac{1+\rho_{max}}{1-\rho_{max}})}\right) = 3 + 8\left(\frac{Z_\alpha}{\ln(\frac{1+\rho' \cdot \hat{p}}{1-\rho' \cdot \hat{p}})}\right). \tag{2}$$

The probability α expresses the likelihood of an attack to be successful.

We evaluate our AES-128 prototype implementation w.r.t. these figures. When an attacker focuses on the storage of an intermediate result in a register, the first round is hard to attack, since all intermediate registers are pre-loaded with random values. Attacking the second round is difficult because of the diffusion property of AES. Therefore, we evaluate the effectiveness of our countermeasure under the assumption that an attacker analyzes the power consumption of the combinatorial logic. In our prototype design, the number of options for the temporal shift depends on which functional block is analyzed. Since SB is a common choice, we evaluate the number of options for the temporal shift of the computation of SB. We take into account that SB and SR can be swapped, which doubles the number of distinct configuration options and leads to $c = 20$. However, since t_c only depends on the number of registers preceding SB and the position of the last register, some configurations lead to the same temporal shift. In fact, there are 8 options with probabilities between 1/20 and 6/20, thus $\hat{p} = 6/20$. Under the conservative assumption of $\rho' = 1$, ρ_{max} decreases to 6/20 and S, the amount of measurements required to break the implementation, increases by a factor of more than 3 for $Z_{alpha} = 0.5$ (the median). This number is not impressive at first sight, but note that this countermeasure can be complemented with for example a masking scheme and that we assumed $\rho' = 1$.

In [10], Clavier *et al.* propose the Sliding Window DPA. Although this attack is smarter since it takes into account what is actually happening in the target device, it is also much more difficult to mount in practice. The basic idea is to jointly analyze several time instants where the target value might occur, therefore effectively reversing the process of de-synchronization. The attack consists of a usual DSCA attack and a postprocessing of the differential traces. Clavier *et al.*

suggest to choose a suitable number of instants with a suitable distance between them to form a "comb". They suggest to slide the comb over each differential trace with a given offset and, at each position, to integrate the trace at all instants selected by the comb. As a result, one obtains the same number of integrated traces as differential traces. The integrated trace corresponding to the correct key guess does not show not a spike, but a clearly visible Gaussian 'peak'. They conclude that if the targeted intermediate result is spread over g consecutive cycles (a cycle is the smallest time unit for a software implementation) their attack requires g times more measurements. It remains, however, unclear how to choose the number of instants or their distance in practice, when one has little knowledge about the device and the implementation. For our prototype with two registers the spreading factor g is 8.

We also note that this countermeasure does not protect the functional blocks but rather the overall architecture.

4.5 Resistance against Fault Analysis

Here we evaluate the level of protection against fault attacks provided by the time jitter countermeasure. Using the definitions introduced in Sect. 3.1, the probability of a successful fault injection is $p = p_{volume} \cdot p_{time} = 6/20$, since we assume $p_{volume} = 1$. However, it is not necessarily possible for the adversary to distinguish between a successful and a non-successful fault injection. This can have a major impact if the fault attack requires multiple successful fault injections and further mathematical cryptanalysis, which is sensitive to incorrect input data. DFA, for example, usually requires several successful fault injections and might sieve out the correct key if a non-successfully faulted cryptogram is amongst the input data. Therefore, the countermeasure is effective although $p = 6/20$ is not that small in this example.

An adversary could also try to inject a fault that alters the circuit's behavior. The effect of a successfully injected fault on the switch matrix would only remain until the next dynamic reconfiguration of the matrix. Since the fault is transient, reconfiguration will bring the circuit back to its normal behavior. A successfully injected fault on any other functional block, on the other hand, remains until system reset. However, since we assume the random fault model and do not consider invasive adversaries it is highly unlikely that an adversary can modify the circuit's behavior in an exploitable way. Nevertheless, the functional blocks can be protected with complementary countermeasures.

We want to mention here one specific attack, though out of the model, that can pose a great risk. Should it be possible to inject a fault that flips a random number of bits in the key register to either zero or one with high probability, an attack as described in [3] can be carried out. Therefore a system designer might want to add further protection. For instance this can be done by duplicating the key register and comparing the contents to the original key register or applying the techniques described in the next section. These countermeasures can in general be applied to the static control part of the design.

5 Spatial and Temporal Jitter

In this section, we protect the cryptographic implementation using both temporal and spatial jitter. To achieve this, not only the moment in time when a value is stored or computed needs to be determined by dynamic reconfiguration, but also the position of the functional blocks. Therefore the time and place when/where a subfunction is executed in the resulting architecture is based on the reconfiguration option.

5.1 Description of a Generic Architecture

In order to further improve the resistance of the implementation against fault analysis attacks, we propose a dynamically reconfigurable architecture in which the location of the subfunctions and intermediate registers is altered randomly based on the output of a TRNG. The general architecture of a system including this countermeasure is depicted in Fig. 6. For each functional block, both a registered and a non-registered variant can be inserted dynamically, depending on the output of the TRNG, causing temporal jitter as described in Sect. 4.1. Moreover, the position of the blocks can be altered depending on a second output of the TRNG, causing spatial jitter. In order to connect the output of the last subfunction in the algorithm to the output of the design, all blocks have an extra output, which is connected to an OR-gate that combines all these extra outputs. Only the last block sends a value to the output, while the other blocks provide the OR-gate with zeros.

Suppose that the order of execution of the subfunctions in the algorithm is fixed and equal to f_0, f_1, ..., f_{n-1}, where f_i is the function that is implemented in block i. Then the number of possible positions of the functional blocks is

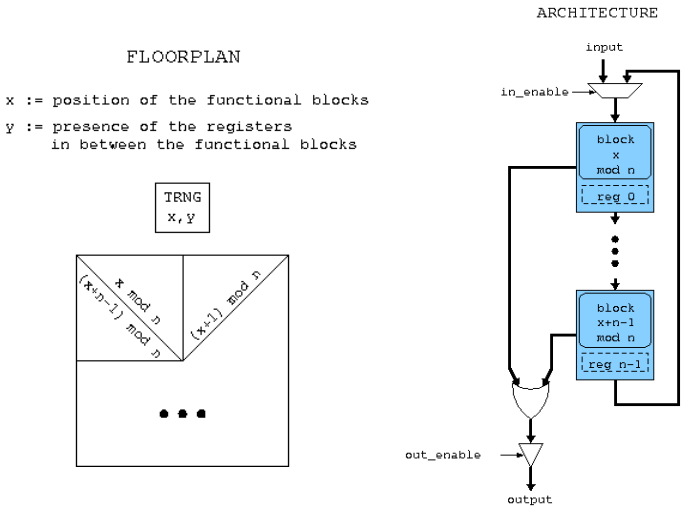

Fig. 6. Modified architecture for improved fault attack resistance

n. The probability of injecting a fault at a certain intended area on the chip surface, as denoted by p_{area} in Sect. 3.1, depends on the technology and the fault injection process. This countermeasure aims in particular at preventing local fault injection processes such as optical fault injection. We assume that the typical spatial focus in such an attack, *e.g.* the focus of a laser, is so small that the probability that a laser fault injection at the same (x,y) coordinates would still affect the same functional block after it has been relocated is negligible. Hence we have $p_{area} = 1/n$.

Suppose that each block can be followed by at most one register. Then the number of possible positions for the intermediate registers is n. The probability of injecting a fault at a certain intended moment in time, as denoted by p_{time} in Sect. 3.1, depends on the targeted subfunction and has a lower bound of $1/(m+1)$.

5.2 Example for AES-128 Encryption

The high-level architecture of our AES-128 prototype is the same as the one in Fig. 2 in Sect. 3.2. The AES coprocessor is implemented according to the general architecture in Fig. 6 with $n = 4$. This means there are four regions which can be configured in eight possible ways, *i.e.* each region can be loaded with ARK, SB, SR, or MC, and each of these functions can be followed by a register. In this case, $p_{area} = 1/4$ and $p_{time} = 1/3$ with $m = 2$. Note that SR and SB can also be interchanged in the round sequence of AES, which increases the number of possible configurations even more. The reconfiguration time for this design is significantly higher than for the one shown in Fig. 4, since more regions need to be reconfigured. The maximal clock frequency on the other hand is similar.

5.3 Can the Countermeasure Be Circumvented?

Again we examine whether the countermeasure can be undone using TA or SPA. The results are similar to those presented in Sect. 4.3. In summary we observed a constant execution time both of the encryption and the partial reconfiguration process and no remarkable patterns in the power traces in either case. Therefore we conclude that neither TA nor SPA enables an adversary to distinguish the configurations with a significant rate of success.

5.4 Resistance against Fault Analysis

Here we evaluate the level of protection against fault attacks provided by the combined spatial and temporal jitter countermeasure. Most of the analysis presented in Sect. 4.5 also applies in this case. However, thanks to the jitter in both domains, the probability of a successful fault injection is $p = p_{volume} \cdot p_{time} = 1/12$.

Another interesting property of the spatial jitter is, that all functional blocks of the cryptosystem are now implemented in reconfigurable areas. Since the effect of a long-lasting transient fault can be undone by dynamic partial reconfiguration, the circuit can effectively recover from an injected fault. For the proposed fault

attack countermeasure, the trade-off between security and reconfiguration delay should be seen in the context of the fault injection frequency. For example: the optical fault injection setup presented in [21] has a maximum laser pulse frequency of 50 Hz. Assuming a fault injection process with this kind of laser, the reconfiguration frequency can be lowered to 50 Hz (plus an additional security margin).

6 Fault Detection

In particular for FPGAs, fault detection can be realized by reading back one, some, or all bitstreams from the reconfigurable areas and comparing them with the reference copy stored in the block RAM. Certain FPGAs already allow to read back bistreams. Comparison to the reference bitstream can be done using (protected) logic gates inside the FPGA. Some vendors provide the stored bitstream with redundant CRC bits. In this case, it is more efficient to examine the bitstream that is read back based on its CRC value. The procedure of reading back the bitstream and comparing it (through logic or CRC check) to the reference copy of the bitstream only detects faults if the reference bitstream cannot be altered in the same way as the bitstream in the reconfigurable area. In our fault model, it is practically infeasible to insert a fault in the reconfigurable area and on the bitstream stored in block RAM with the same effect. Therefore, the probability that this kind of fault detection fails is negligible. Moreover, the scheme can be complemented with traditional fault detection mechanisms such as dual-rail precharge logic with an error state. Another option is to execute the implemented algorithm twice, either in parallel (which doubles the area) or sequentially (which doubles the execution time). In the architectures that we propose, both executions can run in different configurations, which increases the probability of fault detection.

All methods mentioned in this section focus on fault detection. It remains the system designer's choice how to react to an alarm signal. We note that for some attacks outputting the result first and then checking the bitstream for faults might raise an alarm when it is already too late. Checking before outputting seems to be more appropriate in such cases.

7 Conclusion

This paper introduces the use of partial dynamic reconfigurability as a countermeasure against physical attacks. On the one hand, side channel attack resistance can be improved by introducing temporal jitter. On the other hand, fault attack resistance can be improved by introducing spatial and/or temporal jitter. We also suggest a method to add a fault detection mechanism to a reconfigurable hardware design with negligible area overhead.

Acknowledgements

The authors would like to thank Kazuo Sakiyama and Frederik Gommé for their input and support.

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by FWO projects G.0475.05, and G.0300.07, by the European Comission through the IST Programme under Contract IST-2002-507932 ECRYPT NoE, and by the K.U. Leuven-BOF.

The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Ambrose, J.A., Ragel, R.G., Parameswaran, S.: A Smart Random Code Injection to Mask Power Analysis Based Side Channel Attacks. In: Proceedings of CODES+ISSS, pp. 51–56. ACM, New York (2007)
2. Bajard, J.-C., Imbert, L., Liardet, P.-Y., Teglia, Y.: Leak Resistant Arithmetic. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 62–75. Springer, Heidelberg (2004)
3. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
4. Blömer, J., Seifert, J.P.: Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 162–181. Springer, Heidelberg (2003)
5. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Eliminating Errors in Cryptographic Computations. Journal of Cryptology 14(2), 101–119 (2001)
6. Bouesse, G.F., Renaudin, M., Sicard, G.: Improving DPA Resistance of Quasi Delay Insensitive Circuits Using Randomly Time-shifted Acknowledgment Signals. In: da Luz Reis, R.A., Osseiran, A., Pfleiderer, H.J. (eds.) Proceedings of VLSI-SoC. IFIP, vol. 240, pp. 11–24. Springer, Boston (2005)
7. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
8. Ciet, M., Neve, M., Peeters, E., Quisquater, J.-J.: Parallel FPGA Implementation of RSA with Residue Number Systems – Can side-channel threats be avoided? Cryptology ePrint Archive, Report 2004/187 (2004), `http://eprint.iacr.org/`
9. Clavier, C.: Secret External Encodings Do Not Prevent Transient Fault Analysis. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 181–194. Springer, Heidelberg (2007)
10. Clavier, C., Coron, J.S., Dabbous, N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 253–263. Springer, Heidelberg (2000)
11. Handschuh, H., Trichina, E.: Securing Flash Technology. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P. (eds.) Proceedings of FDTC, pp. 3–17. IEEE Computer Society, Los Alamitos (2007)
12. Hemme, L.: A Differential Fault Attack Against Early Rounds of (Triple-)DES. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 254–267. Springer, Heidelberg (2004)

13. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
14. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
15. Lemke-Rust, K., Paar, C.: An Adversarial Model for Fault Analysis Against Low-Cost Cryptographic Devices. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 131–143. Springer, Heidelberg (2006)
16. Maingot, V., Ferron, J.B., Leveugle, R., Pouget, V., Douin, A.: Configuration Errors Analysis in SRAM-based FPGAs: Software Tool and Practical Results. Microelectronics Reliability 47(9-11), 1836–1840 (2007)
17. Mangard, S.: Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 222–235. Springer, Heidelberg (2004)
18. Mesquita, D., Badrignans, B., Torres, L., Sassatelli, G., Robert, M., Moraes, F.: A Cryptographic Coarse Grain Reconfigurable Architecture Robust Against DPA. In: Proceedings of IPDPS, pp. 1–8. IEEE, Los Alamitos (2007)
19. NIST. Advanced Encryption Standard (AES). FIPS Publication 197 (2001)
20. Örs, S.B., Oswald, E., Preneel, B.: Power-Analysis Attacks on an FPGA – First Experimental Results. In: D.Walter, C., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 35–50. Springer, Heidelberg (2003)
21. New Wave Research. Quiklaze ST, `http://www.new-wave.com/1nwrProducts/QuikLaze.htm`
22. Standaert, F.-X., Mace, F., Peeters, E., Quisquater, J.-J.: Updates on the Security of FPGAs Against Power Analysis Attacks. In: Bertels, K., Cardoso, J.M.P., Vassiliadis, S. (eds.) ARC 2006. LNCS, vol. 3985, pp. 335–346. Springer, Heidelberg (2006)
23. Standaert, F.-X., Örs, S.B., Preneel, B.: Power Analysis Attack on an FPGA Implementation of Rijndael: Is Pipelining a DPA Countermeasure?. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 30–44. Springer, Heidelberg (2004)
24. Tillich, S., Herbst, C., Mangard, S.: Protecting AES Software Implementations on 32-bit Processors against Power Analysis. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 141–157. Springer, Heidelberg (2007)
25. Xilinx. OPB HWICAP, `http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_hwicap.pdf`