The First Thorough Side-Channel Hardware Trojan

Maik Ender[1], Samaneh Ghandali[2], Amir Moradi[1], and Christof Paar[1,2]

[1] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{firstname.lastname}@rub.de

[2] University of Massachusetts Amherst, USA
samaneh@umass.edu

Abstract. Hardware Trojans have gained high attention in academia, industry and by government agencies. The effective detection mechanisms and countermeasures against such malicious designs are only possible when there is a deep understanding of how hardware Trojans can be built in practice. In this work, we present a mechanism which shows how easily a stealthy hardware Trojan can be inserted in a provably-secure side-channel analysis protected implementation. Once the Trojan is triggered, the malicious design exhibits exploitable side-channel leakage leading to successful key recovery attacks. Such a Trojan does not add or remove any logic (even a single gate) to the design which makes it very hard to detect. In ASIC platforms, it is indeed inserted by subtle manipulations at the sub-transistor level to modify the parameters of a few transistors. The same is applicable on FPGA applications by changing the routing of particular signals, leading to **null** resource utilization overhead. The underlying concept is based on a secure masked hardware implementation which does not exhibit any detectable leakage. However, by running the device at a particular clock frequency one of the requirements of the underlying masking scheme is not fulfilled anymore, i.e., the Trojan is triggered, and the device's side-channel leakage can be exploited.
Although as a case study we show an application of our designed Trojan on an FPGA-based threshold implementation of the PRESENT cipher, our methodology is a general approach and can be applied on any similar circuit.

1 Introduction

Cryptographic devices are those pieces of (usually) hardware that implement cryptographic algorithm(s) providing different aspects of security. Since such devices often deal with secret information and/or privacy of the users, hardware Trojans have gained high attention in academia and industry as well as government agencies, and can leak the secrets in a particular fashion without the notice of the end users. Indeed, both bodies of research concerning the Trojan design and Trojan detection are large and active. Nevertheless, these two topics are

closely related. The effective detection mechanisms and countermeasures are only possible when there is an understanding of how hardware Trojans can be built.

Amongst several different ways to insert a Trojan into an IC, we can refer to those conducted *i*) by an untrusted semiconductor foundry during manufacturing, *ii*) by the original hardware designer who is pressured by the government bodies, and *iii*) in the third-party IP cores. Most of the hardware Trojans are inserted by modifying a few gates (can be done at different abstraction levels). In short, one of the main goals of the Trojans is to be designed/implemented in such a way that the chance of detection becomes very low. Our focus in this article is those Trojans which leak out the secrets through a side channel. The first such a Trojan has been introduced in [35] and [36] which stealthily leaks out the cryptographic key using leakages through power consumption side channel. The underlying scheme is independent of the cryptographic algorithm and deals only with the secret key. This Trojan, made by a moderately large circuit including an LFSR and leaking circuit, is inserted at the netlist or HDL level. Therefore, it is likely detected by a Trojan inspector. Further, the designs in these works [35, 36] are not parametric Trojans, i.e., they always leak through a side channel, which might be exploited by anybody not only the Trojan attacker.

On the other hand, the cryptographic devices – if pervasive and/or ubiquitous – are in danger of side-channel analysis (SCA) attacks. After around two decades since introduction of such physical attacks [32, 33], integration of dedicated SCA countermeasures is a must for devices which deal with security. Therefore, if the design is not protected against SCA threats, any SCA adversary would be able to reveal the secrets independent of the existence of such a Trojan [36].

In a follow-up work [30], the authors expressed a relatively similar concept on an SCA-protected implementation. Their technique is based on inserting a logical circuit forming an LFRS-based Trojan leaking the internal state of the PRNG. As a side note, random number generators are necessary modules for those SCA-protected implementations which are based on masking [38]. Hence, the Trojan adversary would detect the internal state of the PRNG by means of SCA leakages and can conduct DPA attacks knowing the masks. It should be noted that those products which need to be protected against physical attacks are usually evaluated by a third-party certification body, e.g., through a common criteria evaluation lab. Therefore, due to its relatively large circuit, such a Trojan is very likely detected by an inspector.

As another work in this domain, we should refer to [2], where the Trojan is inserted by changing the dopant polarity of a few transistors in a circuit realized by the DPA-resistant logic style iMDPL [47]. However, none of such logic styles can perfectly provide security, and the leakage of an iMDPL circuit can still be exploited by ordinary SCA adversaries [40].

Our contribution. In short, integrating an SCA Trojan into an SCA-protected design is challenging, if the device is supposed to be evaluated by a third-party certification body. It is because the device should provide the desired SCA protection under a white-box scenario, i.e., all design details including the netlist are known to the evaluation lab. In this work, we present a mechanism to design

a provably- and practically-secure SCA-protected implementation which can be turned into an unprotected implementation by a Trojan adversary. Our Trojan does not add any logic (even a single gate) to the design, making it very hard to detect. In case of ASIC platforms, it is done by slightly changing the characteristic of a few transistors, and for FPGA platforms by changing the routing of particular signals. Most notably, our technique is **not** based on the leakage of the PRNG, and it does not affect the provable-security feature of the underlying design unless the Trojan is triggered. More precisely, our technique leads to inserting a *parametric* Trojan, i.e., under normal condition the device does not exhibit any SCA leakage to be detected by an evaluation lab. By increasing the clock frequency of the malicious device (or by decreasing its supply voltage) the Trojan is triggered and exhibits exploitable leakage. Note that such a high clock frequency is beyond the maximum frequency that the device can correctly operate. Hence, the device is not expected to be evaluated under such a condition by evaluation labs. As we show in the following sections, there is a gap between the maximum clock frequency of the device and the clock frequency where the Trojan is triggered. In other words, by increasing the clock frequency (violating its critical path delay) the device starts to operate faulty; by even more increasing the clock frequency the device operates again correctly while exhibiting SCA leakage (i.e., our inserted Trojan becomes active).

Outline. Section 2 deals with necessary background and definitions in the areas of hardware Trojan and threshold implementation as an SCA countermeasure. Afterwards, in Section 3 we express our core idea how to insert our Trojan into a secure threshold implementation. In Section 4 we give details on how to apply such a technique on a threshold implementation of the PRESENT cipher, and in Section 5 the corresponding result of FPGA-based SCA evaluations are exhibited.

2 Background

2.1 Hardware Trojan

Malicious and intentional modification of integrated circuit (IC) during manufacturing in untrusted foundry is an emerging security concern. This problem exists because the majority of ICs are fabricated abroad, and a government agency can force a foundry to manipulate the design maliciously. Also, an IC designer can be pressured by her own country government to modify the ICs maliciously, e.g., those ICs that are used in overseas products. Another possible insertion point are 3rd party IP cores. In general, a hardware Trojan is a back-door that can be inserted into an integrated circuit as an undesired and malicious modification, which makes the behavior of the IC incorrect.

There are many ways to categorize Trojans such as categorizing based on physical characteristics, design phase, abstraction level, location, triggering mechanism, and functionality. But a common Trojan categorization is based on the activation mechanism (Trojan trigger) and the effect on the circuit functionality (Trojan payload). A set of conditions that cause a Trojan to be activated is

called trigger. Trojans can combinationally or sequentially be triggered. An attacker chooses a rare trigger condition so that the Trojan would not be triggered during conventional design-time verification and manufacturing test. Sequentially-triggered Trojans (time bombs) are activated by the occurrence of a sequence of rare events, or after a period of continuous operation [18].

The goal of the Trojan can be achieved by payload which can change the circuit functionally or leak its secret information. In [27] a categorization method according to how the payload of a Trojan works has been defined; some Trojans after triggering, propagate internal signals to output ports which can reveal secret information to the attackers (explicit payload). Other Trojans may make the circuit malfunction or destroy the whole chip (implicit payload). Another categorization for actions of hardware Trojans has been presented in [59], in which the actions can be categorized into classes of modify functionality, modify specification, leak information, and denial of service.

The work in [29] introduced a manipulation which makes an error detection module to work incorrectly and accept inputs that should be rejected. They showed how a Trojan could be used to change the instruction order in which CPU executes them, leak data by side-channel analysis, and change the content of programmable read only memory. The work in [31] presented how a hardware Trojan, which is inserted into a CPU by adding extra logic into its HDL code, can give an attacker unlimited access to the CPU.

Threats posed by hardware Trojans and the methods of deterring them have been analyzed in [18]. For example a bridging fault by insertion of a resistor and by increasing a net delay by enlarging its capacitance load has been introduced in this work. The works in [19] and [53] discussed about efficient generation of test patterns for hardware Trojans triggered by rare input signals. Hardware Trojans in wireless cryptographic ICs have been discussed in [28]. The goal is to design Trojans to leak secret information through the wireless channel. Detection challenges of such Trojans were discussed in this work and some improvements were proposed based on side-channel signals analysis. The work in [36] proposed a hardware Trojan that leaks the cryptographic key through side channel analysis attack. Similar to the hardware Trojans that were designed as part of a student hardware Trojan challenge at [51], this hardware Trojan was inserted at the netlist or HDL level. The work in [2] presented building stealthy Trojans at the layout-level. A hardware Trojan was inserted into a cryptographically-secure PRNG and into a side-channel resistant Sbox by manipulating the dopant polarity of a few registers. Building hardware Trojans that are triggered by aging was presented in [57]. These Trojans only become active after the IC has been working for a long time.

A class of hardware Trojans – Malicious Off-chip Leakage Enabled by Side-channels (MOLES) – has been presented in [35], which can retrieve secret information through side channels. They formulated the mechanism and detection methods of MOLES in theory and provided a verification process for multi-bit key extractions. A parametric Trojan has been introduced in [34] which triggers with a probability increasing under reduced supply voltage. In [21] a design

methodology for building stealthy parametric hardware Trojans and its application to Bug Attacks [4,5] has been proposed. The Trojans are based on increasing delay of gates of a very rare-sensitized path in a combinatorial circuit, such as an arithmetic multiplier circuit. The Trojans are stealthy and have rare trigger conditions, so that the faulty behavior of the circuit under attack only occurs for very few combinations of the input vectors. Also an attack on the ECDH key agreement protocol by this Trojan has been presented in this work.

2.2 Threshold Implementation

It can be definitely said that *masking* is the most-studied countermeasure against SCA attacks. It is based on the concept of *secret sharing*, where a secret $\boldsymbol{x}$ (e.g., intermediate values of a cipher execution) is represented by a couple of shares $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n)$. In case of an (n, n)-threshold secret sharing scheme, having access to $t < n$ does not reveal any information about $\boldsymbol{x}$. Amongst those is Boolean secret sharing, known as Boolean masking in the context of SCA, where $\boldsymbol{x} = \bigoplus_{i=1}^{n} \boldsymbol{x}^i$. Hence, if the entire computation of a cipher is conduced on such a shared representation, its SCA leakage will be (in average) independent of the secrets as long as no function (e.g., combinatorial circuit) operates on all n shares.

Due to the underlying Boolean construction, application of a linear function $\mathcal{L}(.)$ over the shares is straightforward since $\mathcal{L}(\boldsymbol{x}) = \bigoplus_{i=1}^{n} \mathcal{L}(\boldsymbol{x}^i)$. All the difficulties belong to implementing non-linear functions over such a shared representation. This concept has been applied in hardware implementation of AES (mainly with $n = 2$) with no success [16,39,41,46] until the Threshold Implementation (TI) – based on sound mathematical foundations – has been introduced in [45], which defines minimum number shares $n \geq t + 1$ with t the algebraic degree of the underlying non-linear function. For simplicity (and as our case study is based on) we focus on quadratic Boolean functions, i.e., $t = 2$, and minimum number of shares $n = 3$. Suppose that the TI of the non-linear function $y = \mathcal{F}(\boldsymbol{x})$ is desired, i.e., $(\boldsymbol{y}^1, \boldsymbol{y}^2, \boldsymbol{y}^3) = \mathcal{F}^*(\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3)$, where

$$\boldsymbol{y}^1 \oplus \boldsymbol{y}^2 \oplus \boldsymbol{y}^3 = \mathcal{F}(\boldsymbol{x}^1 \oplus \boldsymbol{x}^2 \oplus \boldsymbol{x}^3).$$

Indeed, each output share $y^{i \in \{1,2,3\}}$ is provided by a component function $\mathcal{F}^i(.,.)$ which receives only two input shares. In other words, one input share is definitely missing in every component function. This, which is a requirement defined by TI as *non-completeness*, supports the aforementioned concept that "no function (e.g., combinatorial circuit) operates on all n shares", and implies the given formula $n \geq t+1$. Therefore, three component functions $\left(\mathcal{F}^1\left(\boldsymbol{x}^2, \boldsymbol{x}^3\right), \mathcal{F}^2\left(\boldsymbol{x}^3, \boldsymbol{x}^1\right), \mathcal{F}^3\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right)\right)$ form the shared output $(\boldsymbol{y}^1, \boldsymbol{y}^2, \boldsymbol{y}^3)$.

Uniformity In order to fulfill the above-given statement that "having access to $t < n$ does not reveal any information about $\boldsymbol{x}$", the shares need to follow a

uniform distribution. For simplicity suppose that $n = 2$, and the shares $(\boldsymbol{x}^1, \boldsymbol{x}^2)$ represent secret $\boldsymbol{x}$. If the distribution of x^1 has a bias (i.e., not uniform) which is known to the adversary, he can observe the distribution of $\boldsymbol{x}^2 = \boldsymbol{x} \oplus \boldsymbol{x}^1$ and guess $\boldsymbol{x}$. Hence, the security of the entire masking schemes[3] relies on the uniformity of the masks. More precisely, when $\boldsymbol{x}^1 = \boldsymbol{m}$, $\boldsymbol{x}^2 = \boldsymbol{x} \oplus \boldsymbol{m}$, and $\boldsymbol{m}$ is taken from a randomness source (e.g., a PRNG), the distribution of $\boldsymbol{m}$ should be uniform (or let say with full entropy).

The same holds for higher-order masking, i.e., $n > 2$. However, not only the distribution of every share but also the joint distribution of every $t < n$ shares is important. In case of $\mathcal{F}^*(.,.,.)$ as a TI of a bijective function $\mathcal{F}(.)$, the *uniformity* property of TI is fulfilled if $\mathcal{F}^*(.,.,.)$ forms a bijection. Otherwise, the security of such an implementation cannot be guaranteed. Note that fulfilling the uniformity property of TI constructions is amongst its most difficult challenges, and it has been the core topic of several articles like [3,9,12,45,48]. Alternatively, the shares can be remasked at the end of every non-uniform shared non-linear function (see [8,42]), which requires a source to provide fresh randomness at every clock cycle. Along the same line, another type of masking in hardware (which reduces the number of shares) has been developed in [23,52], which (almost always) needs fresh randomness to fulfill the uniformity.

We should emphasize that the above given expressions illustrate only the first-order TI of bijective quadratic functions. For the other cases including higher-order TI we refer the interested reader to the original articles [9,12,45].

3 Technique

As explained in former section – by means of TI – it is possible to realize hardware cryptographic devices secure against certain SCA attacks. Our goal is to provide a certain situation that an SCA-secure device becomes insecure while it still operates correctly. Such a dynamic transition from secure to insecure should be available and known only to the Trojan attacker. To this end, we target the uniformity property of a secure TI construction. More precisely, we plan to construct a secure and uniform TI design which becomes non-uniform (and hence insecure) at particular environmental conditions. In order to trigger the Trojan (or let say to provide such a particular environmental conditions) for example we select higher clock frequency than the device maximum operation frequency, or lower power supply than the device nominal supply voltage. It should not be forgotten that under such conditions the underlying device should still maintain its correct functionality.

To realize such a scenario – inspired from the stealthy parametric Trojan introduced in [21] – we intentionally lengthen certain paths of a combinatorial circuit. This is done in such a way that – by increasing the device clock frequency or lowering its supply voltage – such paths become faulty earlier than the other paths. We would achieve our goal if i) the faults cancel each others' effect, i.e.,

[3] Except those which are based on low-entropy masking [17,37].

the functionality of the design is not altered, and *ii*) the design does not fulfill the uniformity property anymore.

In order to explain our technique – for simplicity without loss of generality – we focus on a 3-share TI construction. As explained in Section 2.2 – ignoring the uniformity – achieving a non-complete shared function $\mathcal{F}^*(.,.,.)$ of a given quadratic function $\mathcal{F}(.)$ is straightforward. Focusing on one output bit of $\mathcal{F}(\boldsymbol{x})$, and representing $\boldsymbol{x}$ by s input bits $\langle x_s, \ldots, x_1 \rangle$, we can write

$$\begin{aligned}\mathcal{F}_i(\langle x_s, \ldots, x_1 \rangle) =& k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus \ldots \oplus k_s x_s \oplus \\ & k_{1,2} x_1 x_2 \oplus k_{1,3} x_1 x_3 \oplus \ldots \oplus k_{s-1,s} x_{s-1} x_s.\end{aligned}$$

The coefficients $k_0, \ldots, k_{s-1,s} \in \{0,1\}$ form the Algebraic Normal Form (ANF) of the quadratic function $\mathcal{F}_i : \{0,1\}^s \rightarrow \{0,1\}$. By replacing every input bit x_i by the sum of three corresponding shares $x_i^1 \oplus x_i^2 \oplus x_i^3$, the remaining task is just to split the terms in the ANF to three categories in such a way that each category is independent of one share. This can be done by a method denoted by *direct sharing* [12] as

- $\mathcal{F}_i^1(.,.)$ contains the linear terms x_i^2 and the quadratic terms $x_i^2 x_j^2$ and $x_i^2 x_j^3$.
- $\mathcal{F}_i^2(.,.)$ contains the linear terms x_i^3 and the quadratic terms $x_i^3 x_j^3$ and $x_i^3 x_j^1$.
- $\mathcal{F}_i^3(.,.)$ contains the linear terms x_i^1 and the quadratic terms $x_i^1 x_j^1$ and $x_i^1 x_j^2$.

The same is independently applied on each output bit of $\mathcal{F}(.)$ and all three component functions $\mathcal{F}^1\left(\boldsymbol{x}^2, \boldsymbol{x}^3\right)$, $\mathcal{F}^2\left(\boldsymbol{x}^3, \boldsymbol{x}^1\right)$, $\mathcal{F}^3\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right)$ are constructed that fulfill the non-completeness, but nothing about its uniformity can be said.

There are indeed two different ways to obtain a uniform TI construction:

- If s (the underlying function size) is small, i.e., $s \leq 5$, it can be found that $\mathcal{F}(.)$ is affine equivalent to which s-bit class. More precisely, there is a quadratic class $\mathcal{Q}$ which can represent $\mathcal{F}$ as $\mathcal{A}' \circ \mathcal{Q} \circ \mathcal{A}$ (see [13] for an algorithm to find $\mathcal{A}$ and $\mathcal{A}'$ given $\mathcal{F}$ and $\mathcal{Q}$). A classification of such classes for $s = 3$ and $s = 4$ are shown in [12] and for $s = 5$ in [15]. Since the number of existing quadratic classes are restricted, it can exhaustively be searched to find their uniform TI. Note that while for many quadratic classes the direct sharing (explained above) can reach to a uniform TI, for some quadratic classes no uniform TI exists unless the class is represented by a composition of two other quadratic classes [12]. Supposing that $\mathcal{Q}^*(.,.,.)$ is a uniform TI of $\mathcal{Q}(.)$, applying the affine functions $\mathcal{A}'$ and $\mathcal{A}$ accordingly on each input and output of the component function $\mathcal{Q}^*$ would give a uniform TI of $\mathcal{F}(.)$:

$$\begin{aligned}\mathcal{F}^1(\boldsymbol{x}^2, \boldsymbol{x}^3) =& \mathcal{A}' \circ \mathcal{Q}^1\left(\mathcal{A}\left(\boldsymbol{x}^2\right), \mathcal{A}\left(\boldsymbol{x}^3\right)\right), \\ \mathcal{F}^2(\boldsymbol{x}^3, \boldsymbol{x}^1) =& \mathcal{A}' \circ \mathcal{Q}^2\left(\mathcal{A}\left(\boldsymbol{x}^3\right), \mathcal{A}\left(\boldsymbol{x}^1\right)\right), \\ \mathcal{F}^3(\boldsymbol{x}^1, \boldsymbol{x}^2) =& \mathcal{A}' \circ \mathcal{Q}^3\left(\mathcal{A}\left(\boldsymbol{x}^1\right), \mathcal{A}\left(\boldsymbol{x}^2\right)\right).\end{aligned}$$

This scenario has been followed in several works, e.g., [6, 11, 43, 44, 54].

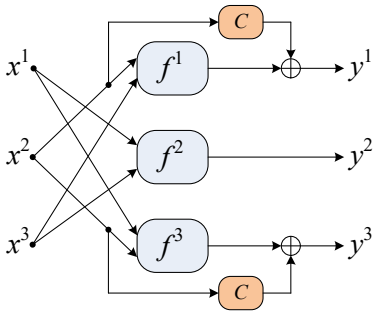


Figure 1: Exemplary TI construction with a correction term C.

– Having a non-uniform TI construction, e.g., obtained by direct sharing, we can add *correction terms* to the component functions in such a way that the correctness and non-completeness properties are not altered, but the uniformity may be achieved. For example, the linear terms x_i^2 and/or the quadratic terms $x_i^2 x_j^2$ as correction terms can be added to the same output bit of **both** component functions $\mathcal{F}^1\left(\boldsymbol{x}^2, \boldsymbol{x}^3\right)$ and $\mathcal{F}^3\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right)$. Addition of any correction term changes the uniformity of the design. Hence, by repeating this process – up to examining all possible correction terms and their combination, which is not feasible for large functions – a uniform construction might be obtained. Such a process has been conducted in [7, 48] to construct uniform TI of PRESENT and Keccak non-linear functions.
We should here refer to a similar approach called remasking [12, 42] where – instead of correction terms – fresh randomness is added to the output of the component functions to make the outputs uniform. In this case, obviously a certain number of fresh mask bits are required at every clock cycle (see [10, 42]).

Our technique is based on the second scheme explained above. If we make the paths related to the correction terms the longest path, by increasing the clock frequency such paths are the first whose delay are violated. As illustrated, each correction term must be added to two component functions (see Figure 1). The paths must be very carefully altered in such a way that the path delay of both instances of the targeted correction term are the longest in the entire design and relatively the same. Hence, at a particular clock frequency both instances of the correction terms are not correctly calculated while all other parts of the design are fault free. This enables the design to still work properly, i.e., it generates correct ciphertext assuming that the underlying design realizes an encryption function. It means that the design operates like an alternative design where no correction terms exists. Hence, the uniformity of the TI construction is not fulfilled and SCA leakage can be exploited. To this end, we should keep a margin between *i*) the path delay of the correction terms and *ii*) the critical path delay of the rest of the circuit, i.e., that of the circuit without correction terms. This

margin guarantees that at a certain high clock frequency the correction terms are canceled out but the critical path delay of the remaining circuit is not violated.

We would like to emphasize that in an implementation of a cipher once one of the TI functions generates non-uniform output (by violating the delay of correction terms), the uniformity is not maintained in the next TI functions and it leads to first-order leakage in all further rounds. If the uniformity is achieved by remasking (e.g., in [24]), the above-expressed technique can have the same effect by making the XOR with fresh mask the longest path. Hence, violating its delay in one TI function would make its output non-uniform, but the fresh randomness may make the further rounds of the cipher again uniform.

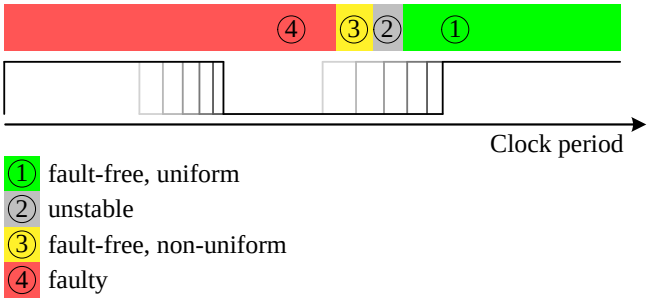


Figure 2: Status of the design with Trojan at different clock frequencies.

Based on Figure 2, which shows a corresponding timing diagram, the device status can be categorized into four states:

- at a low clock frequency (denoted by ①) the device operates fault free and maintains the uniformity,
- by increasing the clock frequency (in the ② period), the circuit first starts to become unstable, when indeed the correction terms do not fully cancel each others' effect, and the hold time and/or setup time of the registers are violated,
- by more increasing the clock frequency (in the ③ period), the delay of both instances of the correction term are violated and the circuit operates fault free, but does not maintain the uniformity, and
- by even more increasing the clock frequency (marked by ④) , the clock period becomes smaller than the critical path delay of the rest of the circuit, and the device does not operate correctly.

The aforementioned margin defines the length of the ② period, which is of crucial importance. If it is very wide, the maximum operation frequency of the resulting circuit is obviously reduced, and the likelihood of the inserted Trojan to be detected by an evaluator is increased.

Correct functionality of the circuit is requited to enable the device being operated in the field. Otherwise, the faulty outputs might be detected (e.g.,

in a communication protocol) and the device may stop operating and prevent collecting SCA traces.

4 Application

In order to show an application of our technique, we focus on a first-order TI design of PRESENT cipher [14] as a case study. The PRESENT Sbox is 4-bit cubic bijection $\mathcal{S}$: `C56B90AD3EF84712`. Hence, its first-order TI needs at least $n = 4$ shares. Alternatively, it can be decomposed to two quadratic bijections $\mathcal{S} : \mathcal{F} \circ \mathcal{G}$ enabling the minimum number of shares $n = 3$ at the cost of having extra register between $\mathcal{F}^*$ and $\mathcal{G}^*$ (i.e., TI of $\mathcal{F}$ and $\mathcal{G}$). As shown in [12], $\mathcal{S}$ is affine equivalent to class $\mathcal{C}_{266}$: `0123468A5BCFED97`, which can be decomposed to quadratic bijections with uniform TI. The works reported in [44, 54, 55] have followed this scenario and represented the PRESENT Sbox as $\mathcal{S} : \mathcal{A}'' \circ \mathcal{Q}' \circ \mathcal{A}' \circ \mathcal{Q} \circ \mathcal{A}$, with many possibilities for the affine functions $\mathcal{A}''$, $\mathcal{A}'$, $\mathcal{A}$ and the quadratic classes $\mathcal{Q}'$ and $\mathcal{Q}$ whose uniform TI can be obtained by direct sharing (see Section 3).

However, the first TI of PRESENT has been introduced in [48], where the authors have decomposed the Sbox by $\mathcal{G}$: `7E92B04D5CA1836F` and $\mathcal{F}$: `08B7A31C46F9ED52`. They have accordingly provided uniform TI of each of such 4-bit quadratic bijections. We focus on this decomposition, and select $\mathcal{G}$ as the target where our Trojan is implemented. Compared to all other related works, we first try to find a **non-uniform** TI of $\mathcal{G}(.)$, and we later make it uniform by means of correction terms. We start with the ANF of $\mathcal{G}(\langle d, c, b, a\rangle) = \langle g_3, g_2, g_1, g_0\rangle$:

$$g_0 = 1 \oplus a \oplus dc \oplus db \oplus cb, \qquad g_2 = 1 \oplus c \oplus b,$$
$$g_1 = 1 \oplus d \oplus b \oplus ca \oplus ba, \qquad g_3 = c \oplus b \oplus a.$$

One possible sharing of $\boldsymbol{y} = \mathcal{G}(\boldsymbol{x})$ can be represented by $\left(\boldsymbol{y}^1, \boldsymbol{y}^2, \boldsymbol{y}^3\right) = \left(\mathcal{G}^1\left(\boldsymbol{x}^2, \boldsymbol{x}^3\right), \mathcal{G}^2\left(\boldsymbol{x}^3, \boldsymbol{x}^1\right), \mathcal{G}^3\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right)\right)$ as

$$y_0^1 = 1 \oplus a^2 \oplus d^2c^3 \oplus d^3c^2 \oplus d^2b^3 \oplus d^3b^2 \oplus c^2b^3 \oplus c^3b^2 \oplus d^2c^2 \oplus d^2b^2 \oplus c^2b^2,$$
$$y_1^1 = 1 \oplus b^2 \oplus d^3 \oplus c^2a^3 \oplus c^3a^2 \oplus b^2a^3 \oplus b^3a^2 \oplus c^2a^2 \oplus b^2a^2,$$
$$y_2^1 = 1 \oplus c^2 \oplus b^2, \qquad y_3^1 = c^2 \oplus b^2 \oplus a^2,$$

$$y_0^2 = a^3 \oplus d^3c^3 \oplus d^1c^3 \oplus d^3c^1 \oplus d^3b^3 \oplus d^1b^3 \oplus d^3b^1 \oplus c^3b^3 \oplus c^1b^3 \oplus c^3b^1,$$
$$y_1^2 = b^3 \oplus d^1 \oplus c^1a^3 \oplus c^3a^1 \oplus b^1a^3 \oplus b^3a^1 \oplus c^3a^3 \oplus b^3a^3,$$
$$y_2^2 = c^3 \oplus b^3, \qquad y_3^2 = c^3 \oplus b^3 \oplus a^3,$$

$$y_0^3 = a^1 \oplus d^1c^1 \oplus d^1c^2 \oplus d^2c^1 \oplus d^1b^1 \oplus d^1b^2 \oplus d^2b^1 \oplus c^1b^1 \oplus c^1b^2 \oplus c^2b^1,$$
$$y_1^3 = b^1 \oplus d^2 \oplus c^1a^2 \oplus c^2a^1 \oplus b^1a^2 \oplus b^2a^1 \oplus c^1a^1 \oplus b^1a^1,$$
$$y_2^3 = c^1 \oplus b^1, \qquad y_3^3 = c^1 \oplus b^1 \oplus a^1,$$

with $\boldsymbol{x}^{i\in\{1,2,3\}} = \langle d^i, c^i, b^i, a^i\rangle$. This is not a uniform sharing of $\mathcal{G}(.)$, and by searching through possible correction terms we found three correction terms c^1b^1, c^2b^2, and c^3b^3 to be added to the second bit of the above-expressed component functions, that lead us to a uniform TI construction. More precisely, by defining

$$\mathcal{C}^1(\boldsymbol{x}^2, \boldsymbol{x}^3) = c^2b^2 \oplus c^3b^3,$$
$$\mathcal{C}^2(\boldsymbol{x}^3, \boldsymbol{x}^1) = c^1b^1 \oplus c^3b^3,$$
$$\mathcal{C}^3(\boldsymbol{x}^1, \boldsymbol{x}^2) = c^1b^1 \oplus c^2b^2,$$

and adding them respectively to y_1^1, y_1^2, and y_1^3, the resulting TI construction becomes uniform. If any of such correction terms is omitted, the uniformity is not maintained. In the following we focus on a single correction term c^2b^2 which should be added to $\mathcal{G}^1(.,.)$ and $\mathcal{G}^3(.,.)$. Note that for the sake of completeness a uniform sharing of $\mathcal{F}$ is given in Appendix A.

4.1 Inserting the Trojan

We realize the Trojan functionality by path delay fault model [58], without modifying the logic circuit. The Trojan is triggered by violating the delay of the combinatorial logic paths that pass through the targeted correction terms c^2b^2. It is indeed a parametric Trojan, which does not require any additional logic. The Trojan is inserted by modifying a few gates during manufacturing, so that their delay increase and add up to the path delay faults.

Given in [21], the underlying method to create a triggerable and stealthy delay-based Trojan consists of two phases: path selection and delay distribution. In the first phase, a set of uniquely-sensitized paths are found that passes through a combinatorial circuit from primary inputs to the primary outputs. Controllability and observability metrics are used to guide the selection of which gates to include in the path to make sure that the path(s) are uniquely sensitized[4]. Furthermore, a SAT-based check is performed to make sure that the path remains sensitizable each time a gate is selected to be added to the path. After a set of uniquely-sensitized paths is selected, the overall delay of the path(s) must be increased so that a delay fault occurs when the path is sensitized. However, any delay added to the gates of the selected path may also cause delay faults on intersecting paths, which would cause undesirable errors and affect the functionality of the circuit. The delay distribution phase addresses this problem by smartly choosing delays for each gate of the selected path to minimize the number of faults caused by intersecting paths. At the same time, the approach ensures that the overall path delay is sufficient for the selected paths to make it faulty.

ASIC Platforms. In an ASIC platform, such Trojans are introduce by slightly modifications on the sub-transistor level so that the parameters of a few transistors

[4]It means that the selected paths are the only ones in the circuit whose critical delay can be violated.

of the design are changed. To increase the delays of transistors in stealthy ways, there are many possible ways in practice. However, such Trojan is very difficult to be detected by e.g., functional testing, visual inspection, and side-channel profiling, because not a single transistor is removed or added to the design and the changes to the individual gates are minor. Also, full reverse-engineering of the IC would unlikely reveal the presence of the malicious manipulation in the design. Furthermore, this Trojan would not present at higher abstraction levels and hence cannot be detected at those levels, because the actual Trojan is inserted at the sub-transistor level.

A path delay fault in a design is sensitized by a sequence of (at least two) consecutive input vectors on consecutive clock cycles. Its reason is charging/discharging of output capacitances of gates of the path. The delay of each gate is determined by its speed in charging or discharging of its output capacitance. Therefore, if the state of the capacitances of gates (belonging to the targeted path) is not changed (i.e., the capacitances do not charge or discharge), the effect of the path delay fault cannot be propagated along the path. Therefore, to trigger the path delay fault, the consecutive input vectors should change the state of the capacitances of the targeted path.

There are several stealthy ways to change slightly the parameters of transistors of a gate and make it slower in charging/discharging its output capacitance (load capacitance). Exemplary, we list three methods below.

Decrease the Width. Usually a standard cell library has different drive strengths for each logic gate type, which correspond to various transistor widths. Current of a transistor is linearly proportional to the transistor width, therefore a transistor with smaller width is slower to charge its load capacitance. One way to increase the delay of a gate is to substitute it with its weaker version in the library which has smaller width, or to create a custom version of the gate with a narrow width, if the lower level information of the gate is available in the library (e.g., SPICE model). The problem here is that an inspector who test the IC optically, may detect the gate downsizing depending on how much the geometry has been changed.

Raise the Threshold. A common way of increasing delay of a gate is to increase the threshold voltage of its transistors by body biasing or doping manipulation. Using high and low threshold voltages at the same time in a design (i.e., Dual-Vt design) is very common approach and provides for designer to have more options to satisfy the speed goals of the design. Devices with low threshold voltage are fast and used where delay is critical; devices with high threshold voltage are slow and used where power consumption is important. Body biasing can change the threshold voltage and hence the delay of a gate through changing the voltage between body and source of the transistor [29]. A reverse body bias in which body is at lower voltage than the source, increases the threshold voltage and makes the device slow. In general, transistors with high threshold voltage will response later when an input switches, and conduct less current. Therefore, the

load capacitances of the transistors will be charged or discharged more slowly. Dopant manipulation and body biasing, are both very difficult to detect.

Increase the Gate Length. Gate length biasing can increase delay of a gate by reducing the current of its transistors [26]. The likelihood of detection of this kind of manipulation depends on the degree of the modification.

FPGA Platforms. In case of the FPGAs, the combinatorial circuits are realized by Look-Up Tables (LUT), in currently-available Xilinx FPGAs, by 6-to-1 or 5-to-2 LUTs and in former generations by 4-to-1 LUTs. The delay of the LUTs cannot be changed by the end users; alternatively we offer the following techniques to make certain paths longer.

Through Switch Boxes. The routings in FPGA devices are made by configuring the switch boxes. Since the switch boxes are made by active components realizing logical switches, a signal which passes through many switch boxes has a longer delay compared to a short signal. Therefore, given a fully placed-and-routed design we can modify the routings by lengthening the selected signals. This is for example feasible by means of `Vivado Design Suite` as a standard tool provided by Xilinx for recent FPGA families and `FPGA Editor` for the older generations. It is in fact needs a high level of expertise, and cannot be done at HDL level. Interestingly, the resulting circuit would not have any additional resource consumption, i.e., the number of utilized LUTs, FFs and Slices, hence hard to detect particularly if the utilization reports are compared.

Through route-thrus LUTs. Alternatively, the LUTs can be configured as logical buffer. This, which is called *route-thrus*, is a usual technique applied by Xilinx tools to enable routing of non-straightforward routes. Inserting a route-thrus LUT into any path, makes its delay longer. Hence, another feasible way to insert Trojans by delay path fault is to introduce as many as required route-thrus LUTs into the targeted path. It should be noted that the malicious design would have more LUT utilization compared to the original design, and it may increase the chance of being detected by a Trojan inspector. However, none of such extra LUTs realizes a logic, and all of them are seen as *route-thrus* LUTs which are very often (almost in any design) inserted by the FPGA vendor's place-and-route tools. Compared to the previous method, this can be done at HDL level (by hard instantiating route-thrus LUTs).

Focusing on our target, i.e., correction term c^2b^2 in $\mathcal{G}^1(.,.)$ and $\mathcal{G}^3(.,.)$, by applying the above-explained procedure, we found the situation which enables introducing delay path fault into such routes:

– Considering Figure 1, the XOR gate which receives the $\mathcal{F}^1$ and $\mathcal{C}$ output should be the last gate in the combinatorial circuit generating y_1^1, i.e., the second bit of $\mathcal{G}^1(.,.)$. The same holds for y_1^3, i.e., the second bit of $\mathcal{G}^3(.,.)$.

– The only paths which should be lengthened are both instances of c^2b^2. Therefore, in case of the FPGA platform we followed both above-explained methods to lengthen such paths, i.e., between i) the output of the LUT generating c^2b^2 and ii) the input of the aforementioned final XOR gate.

We have easily applied the second method (through route-thrus LUTs) at the HDL level by instantiating a couple of LUTs as buffer between the selected path. More detailed results with respect to the number of required route-thrus LUTs and the achieved frequencies to trigger the Trojan are shown in next Section 5. For the first method (through switch boxes) – since our target platform is a Spartan-6 FPGA – we made use of `FPGA Editor` to manually modify the selected routes (see Appendix C for two routes of a signal with different length). We should emphasize that this approach is possible if the correction term c^2b^2 is realized by a unique LUT (can be forced at HDL level by hard instantiating or placing such a module in a deeper hierarchy). Otherwise, the logic generating c^2b^2 might be merged with other logic into a LUT, which avoids having a separate path between c^2b^2 and a LUT that realizes the final XOR gate.

5 Practical Results

5.1 Design Architecture

We made use of the above-explained malicious PRESENT TI Sbox in a design with full encryption functionality. The underlying design is similar to the *Profile 2* of [48], where only one instance of the Sbox is implemented. The nibbles are serially shifted through the state register as well as through the Sbox module while the `PLayer` is performed in parallel in one clock cycle. Following its underlying first-order TI, the 64-bit plaintext is provided by three shares, i.e., second-order Boolean masking, while the 80-bit key is not shared (similar to that of [48] and [10]). Figure 3 shows an overview of the design architecture, which needs 527 clock cycles for a full encryption after the plaintext and key are serially shifted into the state (resp. key) registers.

We should here emphasize that the underlying TI construction is a first-order masking, which can provably provide security against first-order SCA attacks. However, higher-order attacks are expected to exploit the leakage, but they are sensitive to noise [50] since accurately estimating higher-order statistical moments needs huge amount of samples compared to lower-order moments. It is indeed widely known that such masking schemes should be combined with hiding techniques (to reduce the SNR) to practically harden (hopefully disable) the higher-order attacks. As an example we can refer to [44], where a TI construction is implemented by a power-equalization technique. We instead integrated a noise generator module into our target FPGA to increase the noise and hence decrease the SNR. The details of the integrated noise generator module is given in Appendix B. Note that without such a noise generator module, our design would be vulnerable to higher-order attacks and no Trojan would be required to

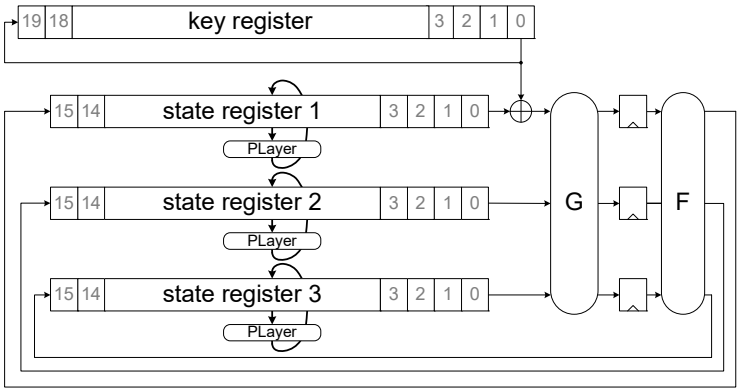


Figure 3: Design architecture of the PRESENT TI as the case study.

reveal the secret. Therefore, the existence of such a hiding countermeasure to make higher-order attacks practically hard is essential.

The design is implemented on a Spartan-6 FPGA board SAKURA-G, as a platform for SCA evaluations [1]. In order to supply the PRESENT core with a high clock frequency, a Digital Clock Manager (DCM) has been instantiated in the target FPGA to multiply the incoming clock by a factor of 8. The external clock was provided by a laboratory adjustable signal generator to enable evaluating the design under different high clock frequencies.

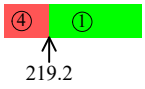
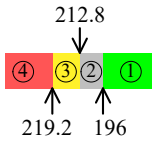
Table 1 shows the resource utilization (excluding the noise generator) as well as the achieved margins for the clock frequency considering *i*) the original design, *ii*) malicious design made by *through switch boxes* method and *iii*) malicious design made by *through route-thrus LUTs* technique. It is noticeable that the first malicious design does not change the utilization figures at all since lengthening the routes are done only through the switch boxes (see Appendix C). Using the second method – in order to achieve the same frequency margins – we added 4 route-thru LUTs (at the HDL level) to each path of the targeted correction term. This led to 8 extra LUT utilization and 4 more Slices; we would like to mention that the combinatorial circuit of the entire TI Sbox (both $\mathcal{G}^*$ $\mathcal{F}^*$) would fit into 29 LUTs (excluding the route-thru ones).

Regarding the frequency ranges, shown in Table 1, it can be seen that the maximum clock frequency of the malicious design is decreased from 219.2 MHz to 196 MHz, i.e., around 10% reduction. However, both ② and ③ periods are very narrow, that makes it hard to be detected either by a Trojan inspector or by an SCA evaluator.

5.2 SCA Evaluations

Measurement Setup. For SCA evaluations we collected power consumption traces (at the Vdd path) of the target FPGA by means of a digital oscilloscope at sampling rate of 1GS/s. It might be thought that when the target design

Table 1: Performance figure of our PRESENT-80 encryption designs.

Design	Method	Utilization				Frequency
		FF	LUT		Slice	
			logic	*route-thrus*		[MHz]
Original	-	299	291	35	226	
Malicious	*switch box*	299	291	35	226	
Malicious	*route-thru LUT*	299	291	43	230	

runs at a high frequency > 150 MHz, such a sampling rate does not suffice to capture all leaking information. However, power consumption traces are already filtered due to the PCB, shunt resistor, measurement setup, etc. Hence, higher sampling rate for such a setting does not improve the attack efficiency[5], and often the bandwidth of the oscilloscope is even manually limited for noise reduction purposes (see [49]).

Methodology. In order to examine the SCA resistance of our design(s) in both settings, i.e., whether the inserted Trojan is triggered or not, we conducted two evaluation schemes. We first performed non-specific t-test (fixed versus random) [22, 56] to examine the existence of detectable leakage. Later in case where the Trojan is triggered, we also conduct key-recovery attacks.

It should be mentioned that both of our malicious designs (see Table 1) operate similarly. It means that when the Trojan is triggered, the evaluation of both designs led to the same results. Therefore, below we exemplary show the result of the one formed by *through route-thrus LUTs*.

To validate the setup, we start with a non-specific t-test when the PRNG of the target design (used to share the plaintext for the TI PRESENT encryption) is turned off, i.e., generating always zero instead of random numbers. To this end, we collected 100,000 traces when the design is operated at 168 MHz, i.e., the Trojan is not triggered. We followed the concept given in [56] for the collection of traces belonging to fixed and random inputs. The result of the t-test (up to third-order) is shown in Figure 4, confirming the validity of the setup and the developed evaluation tools.

[5] It is not the case for EM-based analyses.

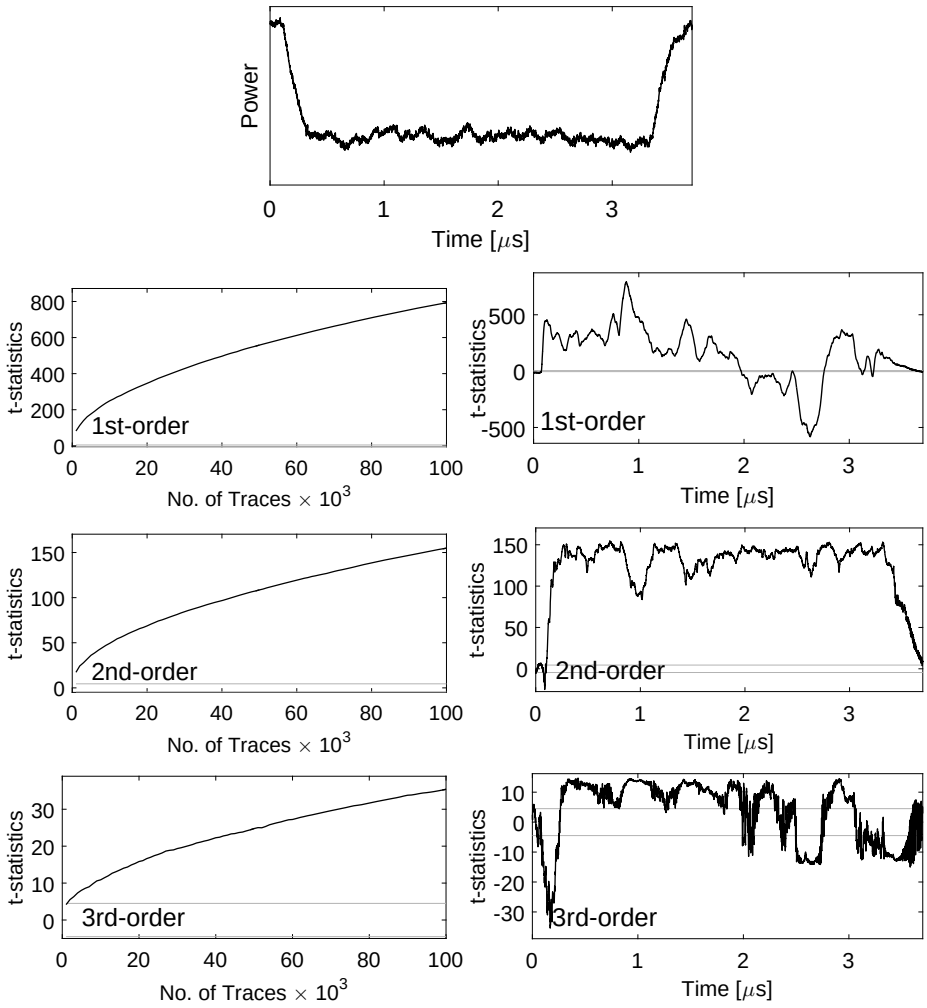


Figure 4: PRNG off, clock 168 MHz (Trojan not triggered), (top) a sample power trace, t-test results (right) with 100,000 traces, (left) absolute maximum over the number of traces.

To repeat the same process when the PRNG is turned on, i.e., the masks for initial sharing of the plaintext are uniformly distributed, we collected 100,000,000 traces for non-specific t-test evaluations. In this case, the device still operates at 168 MHz, i.e., the Trojan is not triggered. The corresponding results are shown in Figure 5. Although the underlying design is a realization of a first-order secure TI, it can be seen from the presented results that second- and third-order leakages are also not detectable. As stated before, this is due to the integration of the noise generator module which affects the detectability of higher-order leakages (see Appendix B).

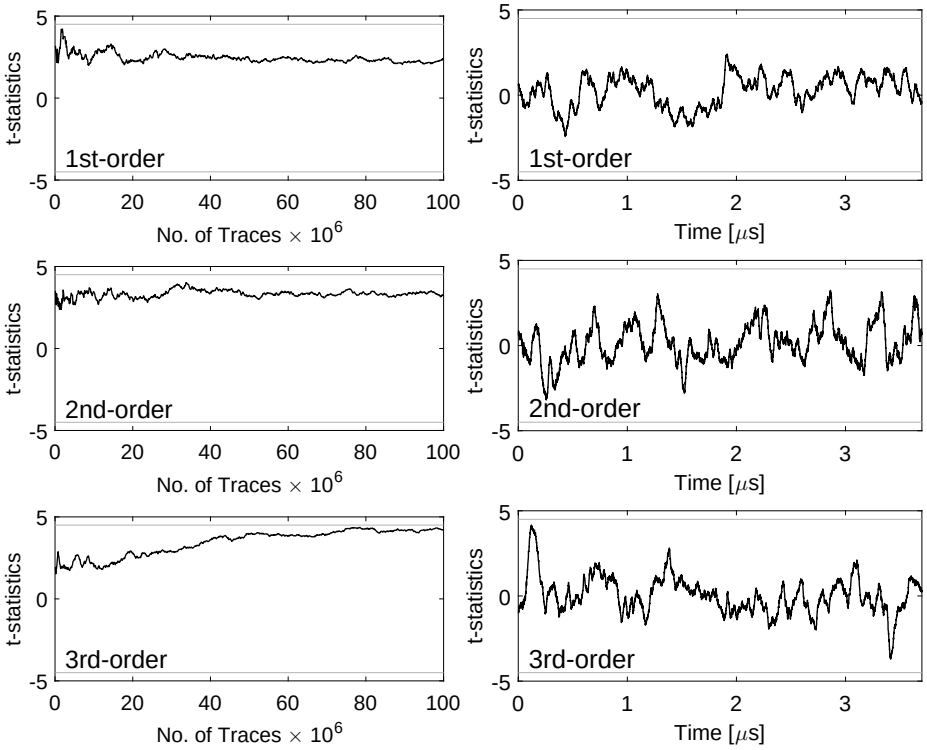


Figure 5: PRNG on, clock 168 MHz (Trojan not triggered), t-test results (right) with 100,000,000 traces, (left) absolute maximum over the number of traces.

As the last step, the same scenario is repeated when the clock frequency is increased to 216 MHz, where the design is in the ③ period, i.e., with correct functionality and without uniformity. Similar to the previous experiment, we collected 100,000,000 traces for a non-specific t-test, whose results are shown in Figure 6. As shown by the graphics, there is detectable leakage through all statistical moments but with lower t-statistics compared to the case with PRNG off. Therefore, we have also examine the feasibility of key recovery attacks. To this end, we made use of those collected traces which are associated with random inputs, i.e., around 50,000,000 traces of the last non-specific t-test. We conducted several different CPA and DPA attacks considering intermediate values of the underlying PRESENT encryption function. The most successful attack was recognized as classical DPA attacks [33] targeting a key nibble by predicting an Sbox output bit at the first round of the encryption. As an example, Figure 7 presents an exemplary corresponding result.

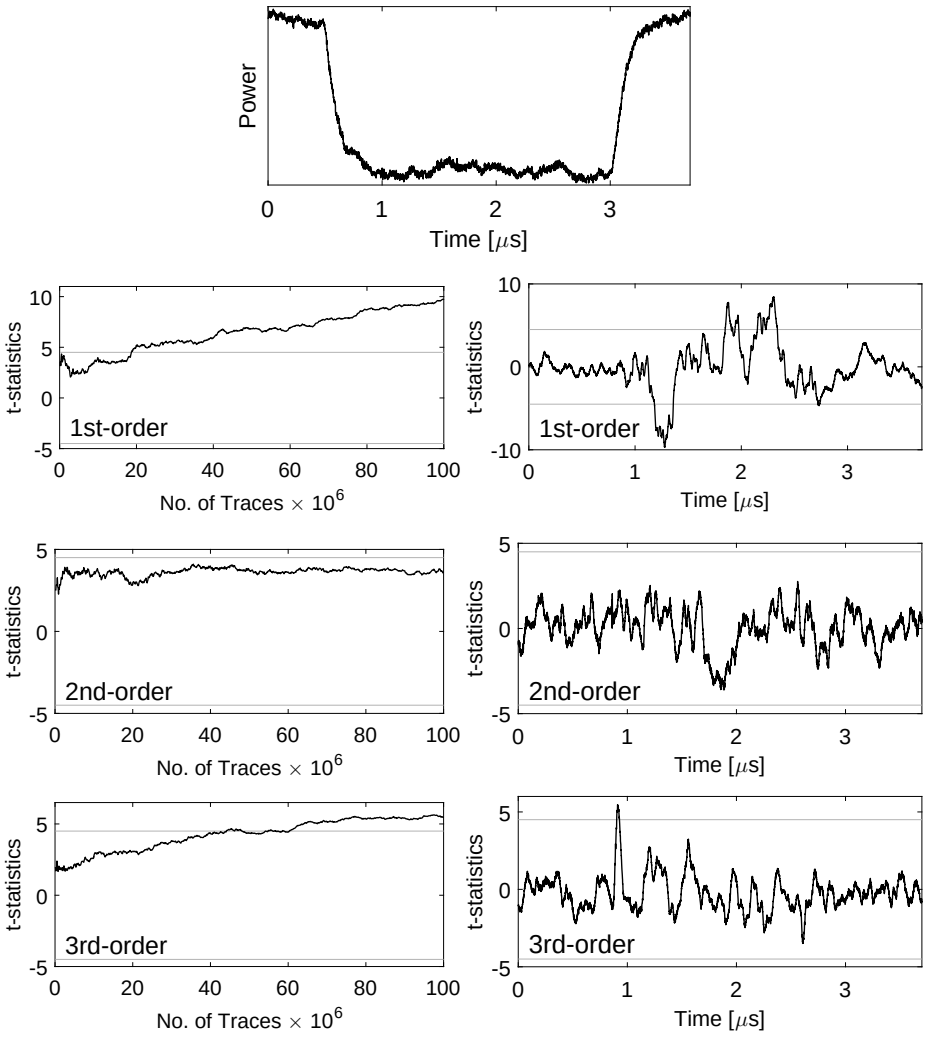


Figure 6: PRNG on, clock 216 MHz (Trojan triggered), (top) a sample power trace, t-test results (right) with 100,000,000 traces, (left) absolute maximum over the number of traces.

6 Conclusions

In this work it is shown how to insert a parametric hardware Trojan with very low overhead into SCA-resistance designs. The presented Trojan is capable of being integrated into both ASIC and FPGA platforms. Since it does not add any logic into the design (particularly its resource utilization in FPGAs can be null), the chance of being detected is expected to be very low. Compared to the

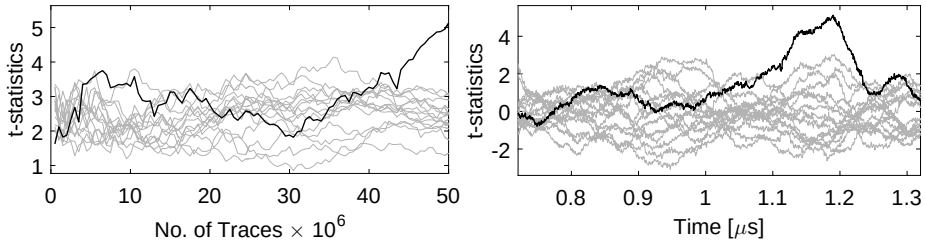


Figure 7: PRNG on, clock 216 MHz (Trojan triggered), 50,000,000 traces, DPA attack result targeting a key nibble based on an Sbox output bit at the first round.

original design, its only footprint is around 10% decrease in the maximum clock frequency.

We have shown that by increasing the clock frequency, the malicious threshold implementation design starts leaking exploitable information through side channels. Hence, the Trojan adversary can trigger the Trojan and make use of the exploitable leakage, while the design can pass SCA evaluations when the Trojan is not triggered. More precisely, suppose that the maximum clock frequency of the malicious device is 196 MHz. Hence, in an evaluation lab its SCA leakage will not be examined at 200 MHz because the device does not operate correctly. However, the Trojan adversary runs the device at 216 MHz and the SCA leakage becomes exploitable. To the best of our knowledge, compared to the previous works in the areas of side-channel hardware Trojans, our construction is the only one which is applied on a provably-secure SCA countermeasure, and is parametric with very low overhead.

A raising question is whether a control over the clock frequency by the Trojan adversary is a practical assumption. Such a control is usually available in FPGA designs since they are mainly externally clocked, and internally multiplied by PLL or DCM. In ASIC or embedded designs, the clock is more often generated internally, hence no control. Nevertheless, by decreasing the supply voltage the same effect can be seen. It can also be criticized that when the attacker has control over the clock, fault-injection attacks by clock glitch can also be a threat. As a message of this paper, overclocking and – at the same time – power supply reduction should be internally monitored to avoid such an SCA-based Trojan being activated. Related to this topic we should refer to [20], where the difficulties of embedding a "clock frequency monitor" in presence of supply voltage changes are shown.

Acknowledgments

The work was partially funded through grants ERC Advanced 695022 and NSF CNS-1421352.

A Uniform TI of $\mathcal{F}$

Considering $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x})$ and $\boldsymbol{x}^{i\in\{1,2,3\}} = \langle d^i, c^i, b^i, a^i\rangle$ – derived by direct sharing – we present one of its uniform sharing $\left(\boldsymbol{y}^1, \boldsymbol{y}^2, \boldsymbol{y}^3\right) = \left(\mathcal{F}^1\left(\boldsymbol{x}^2, \boldsymbol{x}^3\right), \mathcal{F}^2\left(\boldsymbol{x}^3, \boldsymbol{x}^1\right), \mathcal{F}^3\left(\boldsymbol{x}^1, \boldsymbol{x}^2\right)\right)$ as

$$
\begin{aligned}
y_0^1 &= b^2 \oplus c^2a^2 \oplus c^2a^3 \oplus c^3a^2,\\
y_1^1 &= c^2 \oplus b^2 \oplus d^2a^2 \oplus d^2a^3 \oplus d^3a^2,\\
y_2^1 &= d^2 \oplus b^2a^2 \oplus b^2a^3 \oplus b^3a^2,\\
y_3^1 &= c^2 \oplus b^2 \oplus a^2 \oplus d^2a^2 \oplus d^2a^3 \oplus d^3a^2,
\end{aligned}
$$

$$
\begin{aligned}
y_0^2 &= b^3 \oplus c^3a^3 \oplus c^1a^3 \oplus c^3a^1,\\
y_1^2 &= c^3 \oplus b^3 \oplus d^3a^3 \oplus d^1a^3 \oplus d^3a^1,\\
y_2^2 &= d^3 \oplus b^3a^3 \oplus b^1a^3 \oplus b^3a^1,\\
y_3^2 &= c^3 \oplus b^3 \oplus a^3 \oplus d^3a^3 \oplus d^1a^3 \oplus d^3a^1,
\end{aligned}
$$

$$
\begin{aligned}
y_0^3 &= b^1 \oplus c^1a^1 \oplus c^1a^2 \oplus c^2a^1,\\
y_1^3 &= c^1 \oplus b^1 \oplus d^1a^1 \oplus d^1a^2 \oplus d^2a^1,\\
y_2^3 &= d^1 \oplus b^1a^1 \oplus b^1a^2 \oplus b^2a^1,\\
y_3^3 &= c^1 \oplus b^1 \oplus a^1 \oplus d^1a^1 \oplus d^1a^2 \oplus d^2a^1.
\end{aligned}
$$

B Noise Generator

We have built a noise generator as an independent module, i.e., it does not have any connection to the target PRESENT design and operates independently. We followed one the concepts introduced in [25]. As shown by Figure 8, it is made as a combination of a ring oscillator, an LFSR, and several shift registers. The actual power is consumed by the shift registers. Every shift register instantiates a `SRLC32E` primitive, which is a 32-bit shift register within a single LUT inside a `SLICEM`. The shift registers are initialized with the consecutive values of 01. Every shift register's output is feedback to its input and shifted by one at every clock cycle when enabled. Thus, every shift operation toggles the entire bits inside the registers, which maximizes the power consumption of the shift register.

The ring oscillator, made of 31 inverter LUTs, acts as the clock source inside the noise module for both the LFSR and the shift registers. The LFSR realizes the irreducible polynomial $x^{19} + x^{18} + x^{17} + x^{14} + 1$ to generate a pseudo-random clock enable signal for the shift registers.

We instantiated 4×8 instances of the shift register LUTs, fitting into 8 Slices. The ring oscillator required 17 Slices (as stated, made of 31 inverters), and the LFSR fits into 2 Slices, made by 1 LUT for the feedback function, 2 FFs and 2 shift register LUTs. Overall, the entire independent noise generator module required 27 Slices.

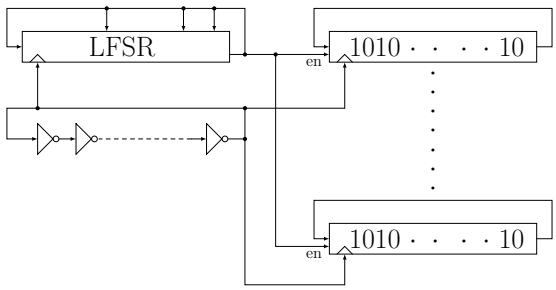


Figure 8: Block diagram of the noise generator.

C Different Routings in FPGA

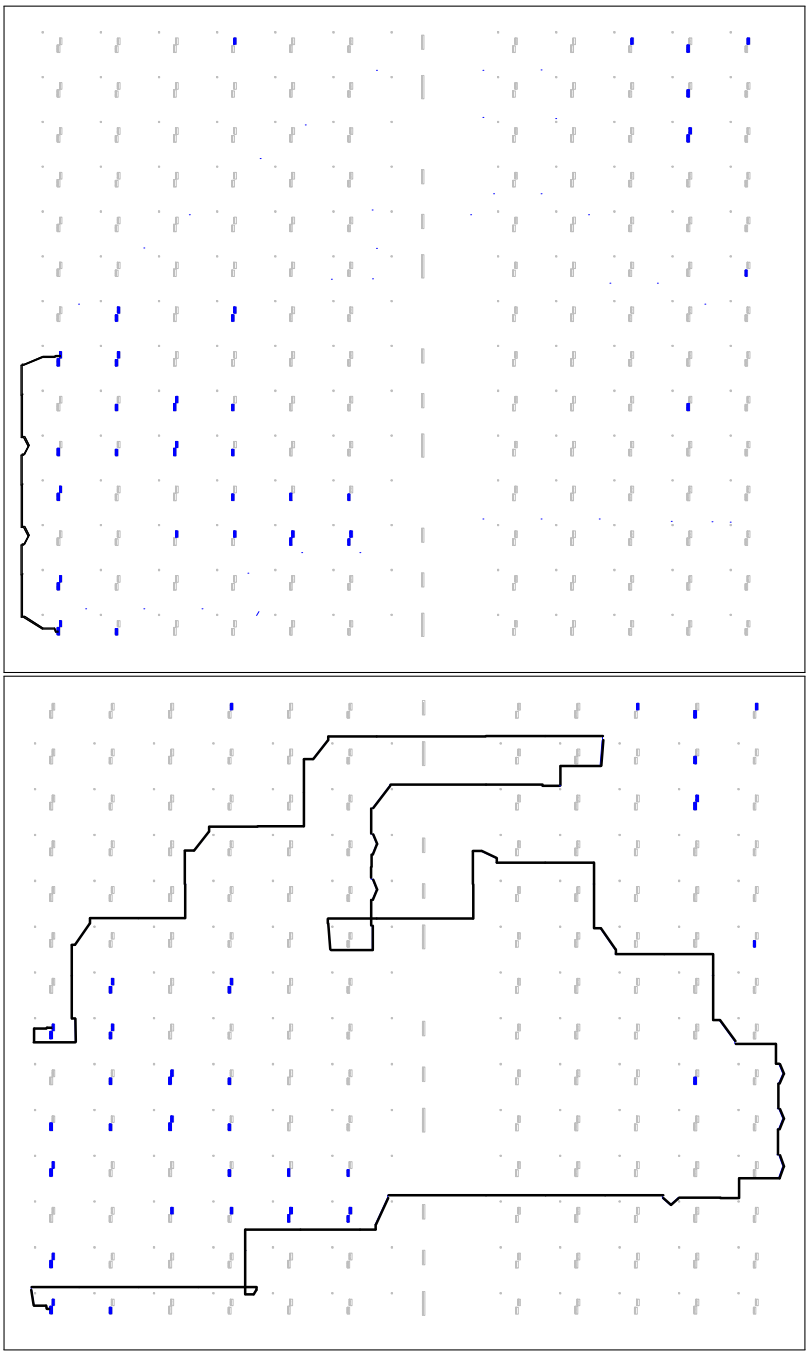


Figure 9: Two routes of the same signal in a Spartan-6 FPGA, manually perfromed by `FPGA Editor`.

References

1. : Side-channel AttacK User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html
2. Becker, G.T., Regazzoni, F., Paar, C., Burleson, W.P.: Stealthy Dopant-Level Hardware Trojans. In: CHES 2013. Volume 8086 of Lecture Notes in Computer Science., Springer (2013) 197–214
3. Beyne, T., Bilgin, B.: Uniform First-Order Threshold Implementations. In: SAC 2016. Lecture Notes in Computer Science, Springer (2017) to appear, eprint.iacr.org/2016/715.pdf.
4. Biham, E., Carmeli, Y., Shamir, A.: Bug Attacks. In: CRYPTO 2008. Volume 5157 of Lecture Notes in Computer Science., Springer (2008) 221–240
5. Biham, E., Carmeli, Y., Shamir, A.: Bug attacks. Journal of Cryptology **29**(4) (2016) 775–805
6. Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In: CHES 2013. Volume 8086 of Lecture Notes in Computer Science., Springer (2013) 142–158
7. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Assche, G.V.: Efficient and First-Order DPA Resistant Implementations of Keccak. In: CARDIS 2013. Volume 8419 of Lecture Notes in Computer Science., Springer (2014) 187–199
8. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A More Efficient AES Threshold Implementation. In: AFRICACRYPT 2014. Volume 8469 of Lecture Notes in Computer Science., Springer (2014) 267–284
9. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: ASIACRYPT 2014. Volume 8874 of Lecture Notes in Computer Science., Springer (2014) 326–343
10. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-Offs for Threshold Implementations Illustrated on AES. IEEE Trans. on CAD of Integrated Circuits and Systems **34**(7) (2015) 1188–1200
11. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold Implementations of All 3×3 and 4×4 S-Boxes. In: CHES 2012. Volume 7428 of Lecture Notes in Computer Science., Springer (2012) 76–91
12. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N., Vitkup, V.: Threshold Implementations of Small S-boxes. Cryptography and Communications **7**(1) (2015) 3–33
13. Biryukov, A., Cannière, C.D., Braeken, A., Preneel, B.: A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In: EUROCRYPT 2003. Volume 2656 of Lecture Notes in Computer Science., Springer (2003) 33–50
14. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: CHES 2007. Volume 4727 of Lecture Notes in Computer Science., Springer (2007) 450–466
15. Bozilov, D., Bilgin, B., Sahin, H.A.: A Note on 5-bit Quadratic Permutations' Classification. IACR Trans. Symmetric Cryptol. **2017**(1) (2017) 398–404
16. Canright, D., Batina, L.: A Very Compact "Perfectly Masked" S-Box for AES. In: ACNS 2008. Volume 5037 of Lecture Notes in Computer Science. (2008) 446–459
17. Carlet, C., Danger, J., Guilley, S., Maghrebi, H.: Leakage Squeezing of Order Two. In: INDOCRYPT 2012. Volume 7668 of Lecture Notes in Computer Science., Springer (2012) 120–139

18. Chakraborty, R.S., Narasimhan, S., Bhunia, S.: Hardware Trojan: Threats and emerging solutions. In: HLDVT 2009, IEEE Computer Society (2009) 166–171
19. Chakraborty, R.S., Wolff, F.G., Paul, S., Papachristou, C.A., Bhunia, S.: MERO: A Statistical Approach for Hardware Trojan Detection. In: CHES 2009. Volume 5747 of Lecture Notes in Computer Science., Springer (2009) 396–410
20. Endo, S., Li, Y., Homma, N., Sakiyama, K., Ohta, K., Fujimoto, D., Nagata, M., Katashita, T., Danger, J., Aoki, T.: A Silicon-Level Countermeasure Against Fault Sensitivity Analysis and Its Evaluation. IEEE Trans. VLSI Syst. **23**(8) (2015) 1429–1438
21. Ghandali, S., Becker, G.T., Holcomb, D., Paar, C.: A Design Methodology for Stealthy Parametric Trojans and Its Application to Bug Attacks. In: CHES 2016. Volume 9813 of Lecture Notes in Computer Science., Springer (2016) 625–647
22. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance validation. In: NIST non-invasive attack testing workshop. (2011) http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.
23. Gross, H., Mangard, S., Korak, T.: An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In: CT-RSA 2017. Volume 10159 of Lecture Notes in Computer Science., Springer (2017) 95–112
24. Groß, H., Wenger, E., Dobraunig, C., Ehrenhöfer, C.: Suit up! - Made-to-Measure Hardware Implementations of ASCON. In: DSD 2015, IEEE Computer Society (2015) 645–652
25. Güneysu, T., Moradi, A.: Generic Side-Channel Countermeasures for Reconfigurable Devices. In: CHES 2011. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 33–48
26. Gupta, P., Kahng, A.B., Sharma, P., Sylvester, D.: Gate-length biasing for runtime-leakage control. IEEE Trans. on CAD of Integrated Circuits and Systems **25**(8) (2006) 1475–1485
27. Jin, Y., Makris, Y.: Hardware Trojan Detection Using Path Delay Fingerprint. In: HOST 2008, IEEE Computer Society (2008) 51–57
28. Jin, Y., Makris, Y.: Hardware Trojans in Wireless Cryptographic ICs. IEEE Design & Test of Computers **27**(1) (2010) 26–35
29. Karri, R., Rajendran, J., Rosenfeld, K., Tehranipoor, M.: Trustworthy Hardware: Identifying and Classifying Hardware Trojans. IEEE Computer **43**(10) (2010) 39–46
30. Kasper, M., Moradi, A., Becker, G.T., Mischke, O., Güneysu, T., Paar, C., Burleson, W.: Side channels as building blocks. J. Cryptographic Engineering **2**(3) (2012) 143–159
31. King, S.T., Tucek, J., Cozzie, A., Grier, C., Jiang, W., Zhou, Y.: Designing and Implementing Malicious Hardware. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET 2008, USENIX Association (2008)
32. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: CRYPTO 1996. Volume 1109 of Lecture Notes in Computer Science., Springer (1996) 104–113
33. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: CRYPTO 1999. Volume 1666 of Lecture Notes in Computer Science., Springer (1999) 388–397
34. Kumar, R., Jovanovic, P., Burleson, W.P., Polian, I.: Parametric Trojans for Fault-Injection Attacks on Cryptographic Hardware. In: FDTC 2014, IEEE Computer Society (2014) 18–28
35. Lin, L., Burleson, W., Paar, C.: MOLES: Malicious off-chip leakage enabled by side-channels. In: ICCAD 2009, ACM (2009) 117–122

36. Lin, L., Kasper, M., Güneysu, T., Paar, C., Burleson, W.: Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering. In: CHES 2009. Volume 5747 of Lecture Notes in Computer Science., Springer (2009) 382–395
37. Maghrebi, H., Guilley, S., Danger, J.: Leakage Squeezing Countermeasure against High-Order Attacks. In: WISTP 2011. Volume 6633 of Lecture Notes in Computer Science., Springer (2011) 208–223
38. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (2007)
39. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: CHES 2005. Volume 3659 of Lecture Notes in Computer Science., Springer (2005) 157–171
40. Moradi, A., Kirschbaum, M., Eisenbarth, T., Paar, C.: Masked Dual-Rail Precharge Logic Encounters State-of-the-Art Power Analysis Methods. IEEE Trans. VLSI Syst. **20**(9)
41. Moradi, A., Mischke, O., Eisenbarth, T.: Correlation-Enhanced Power Analysis Collision Attack. In: CHES 2010. Volume 6225 of Lecture Notes in Computer Science., Springer (2010) 125–139
42. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: EUROCRYPT 2011. Volume 6632., Springer (2011) 69–88
43. Moradi, A., Schneider, T.: Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori. In: ASIACRYPT 2016. Volume 10031 of Lecture Notes in Computer Science. (2016) 517–547
44. Moradi, A., Wild, A.: Assessment of Hiding the Higher-Order Leakages in Hardware - What Are the Achievements Versus Overheads? In: CHES 2015. Volume 9293 of Lecture Notes in Computer Science., Springer (2015) 453–474
45. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. J. Cryptology **24**(2) (2011) 292–321
46. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-Box. In: FSE 2005. Volume 3557 of Lecture Notes in Computer Science., Springer (2005) 413–423
47. Popp, T., Kirschbaum, M., Zefferer, T., Mangard, S.: Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In: CHES 2007. Volume 4727 of Lecture Notes in Computer Science., Springer (2007) 81–94
48. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-Channel Resistant Crypto for Less than 2, 300 GE. J. Cryptology **24**(2) (2011) 322–345
49. Pozo, S.M.D., Standaert, F.: Getting the Most Out of Leakage Detection - Statistical tools and Measurement Setups Hand in Hand. In: COSADE 2017. Lecture Notes in Computer Science, Springer (2017) to appear.
50. Prouff, E., Rivain, M., Bevan, R.: Statistical Analysis of Second Order Differential Power Analysis. IEEE Trans. Computers **58**(6) (2009) 799–811
51. Rajendran, J., Jyothi, V., Karri, R.: Blue team red team approach to hardware trust assessment. In: ICCD 2011, IEEE Computer Society (2011) 285–288
52. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: CRYPTO 2015. Volume 9215 of Lecture Notes in Computer Science., Springer (2015) 764–783
53. Saha, S., Chakraborty, R.S., Nuthakki, S.S., Anshul, Mukhopadhyay, D.: Improved Test Pattern Generation for Hardware Trojan Detection Using Genetic Algorithm and Boolean Satisfiability. In: CHES 2015. Volume 9293 of Lecture Notes in Computer Science., Springer (2015) 577–596

54. Sasdrich, P., Moradi, A., Güneysu, T.: Affine Equivalence and Its Application to Tightening Threshold Implementations. In: SAC 2015. Volume 9566 of Lecture Notes in Computer Science., Springer (2015) 263–276
55. Sasdrich, P., Moradi, A., Güneysu, T.: Hiding Higher-Order Side-Channel Leakage - Randomizing Cryptographic Implementations in Reconfigurable Hardware. In: CT-RSA 2017. Volume 10159 of Lecture Notes in Computer Science., Springer (2017) 131–146
56. Schneider, T., Moradi, A.: Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In: CHES 2015. Volume 9293 of Lecture Notes in Computer Science., Springer (2015) 495–513
57. Shiyanovskii, Y., Wolff, F.G., Rajendran, A., Papachristou, C.A., Weyer, D.J., Clay, W.: Process reliability based trojans through NBTI and HCI effects. In: Adaptive Hardware and Systems AHS 2010, IEEE (2010) 215–222
58. Smith, G.L.: Model for Delay Faults Based upon Paths. In: International Test Conference 1985, IEEE Computer Society (1985) 342–351
59. Wang, X., Salmani, H., Tehranipoor, M., Plusquellic, J.F.: Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis. In: DFT 2008, IEEE Computer Society (2008) 87–95