

# Side-Channel Hardware Trojan for Provably-Secure SCA-Protected Implementations

Samaneh Ghandali, Thorben Moos, Amir Moradi, Christof Paar, *Fellow, IEEE*

**Abstract**—Hardware Trojans have drawn the attention of academia, industry and government agencies. Effective detection mechanisms and countermeasures against such malicious designs can only be developed when there is a deep understanding of how hardware Trojans can be built in practice, in particular Trojans specifically designed to avoid detection. In this work, we present a mechanism to introduce an extremely stealthy hardware Trojan into cryptographic primitives equipped with provably-secure first-order side-channel countermeasures. Once the Trojan is triggered, the malicious design exhibits exploitable side-channel leakage, leading to successful key recovery attacks. Generally, such a Trojan requires neither addition nor removal of any logic which makes it extremely hard to detect. On ASICs, it can be inserted by subtle manipulations at the sub-transistor level and on FPGAs by changing the routing of particular signals, leading to zero logic overhead. The underlying concept is based on modifying a securely-masked hardware implementation in such a way that running the device at a particular clock frequency violates one of its essential properties, leading to exploitable leakage. We apply our technique to a Threshold Implementation of the PRESENT block cipher realized in two different CMOS technologies, and show that triggering the Trojan makes the ASIC prototypes vulnerable.

**Index Terms**—Hardware Trojan, Threshold Implementation, Side-Channel Analysis (SCA), PRESENT, ASIC.

## I. INTRODUCTION

Cryptographic primitives are often the most trusted components in modern security solutions, ranking from network routers to IoT devices. Unfortunately, this makes cryptographic algorithms an attractive target for subversion by malicious actors. Manipulating hardware implementations as opposed to software implementations can lead to cryptographic Trojans that are particularly difficult to detect. It is widely believed that such Trojans are of special interest to nation-state adversaries.

Hardware Trojans have moved in the focus of academia, industry and government agencies over the last decade. In particular, Trojan detection has become an active research area. At the same time, the development of effective detection mechanisms and countermeasures require a thorough understanding of how hardware Trojans can be built. This contribution is concerned with cryptographic Trojans which possess zero overhead in terms of logic resources and are thus, extremely stealthy.

There are several paths to introduce a Trojan into an IC during the design cycle. Three general approaches are: insertion (*i*) by an untrusted semiconductor foundry during

manufacturing, (*ii*) by the original hardware designer, possibly pressured by a government body or through subverted design tools, and (*iii*) through third-party IP cores. Most hardware Trojans require the modification or insertion of additional logic resources (which can be done at different abstraction levels). In most cases, adversaries attempt to design and implement Trojans in such a way that the chance of detection becomes very low. Our focus in this contribution are Trojans which disclose crucial secrets through side channels. The first such Trojan has been introduced in [1], [2], which stealthily leaks out the cryptographic key through a power side channel. The underlying Trojan construction is independent of the cryptographic algorithm and is focused on key-leakage. The Trojan, based on a moderately large circuit including an LFSR and leaking circuit, is inserted at the netlist or HDL level. Such Trojans, however, can be detected through standard VLSI analysis techniques, e.g., through imaging-based reverse engineering. Another attack vector for hardware Trojans is the subversion of side-channel countermeasures. Cryptographic implementations are often threatened by side-channel analysis (SCA). Two decades after the introduction of SCA attacks [3], [4], integration of dedicated countermeasures is a must in many applications. In a follow-up work to reference [2], a related concept that subverts an SCA-protected implementation is introduced [5]. The technique is based on inserting a logical circuit forming an LFSR-based Trojan, which leaks the internal state of the PRNG used for masking. The adversary can now detect the internal state of the PRNG by means of SCA leakages, and can conduct DPA attacks based on her knowledge of the mask. It should be noted that products which need to be protected against physical attacks are often 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 will likely be detected by an inspector. Another relevant prior work is reference [6], where a Trojan is inserted by changing the dopant polarity of a few transistors in a circuit that realizes the DPA-resistant logic style iMDPL [7]. However, iMPDL (and related logic styles) do not provide perfect SCA protection, and the leakage of an iMDPL circuit can still be exploited by ordinary SCA adversaries, even without a Trojan [8]. In reference [9] analog and RF Trojans are introduced, which can detect an extremely rare sequential event as a trigger with just a handful of transistors added to the circuit. This poses a real threat to the IC supply chain. However, since such analog Trojans are triggered by high frequency wire-flops in the processor, abnormal toggling detection methods may detect them [10].

S. Ghandali is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA,01003 USA  
E-mail: samaneh@umass.edu.

T. Moos, A. Moradi and C. Paar are with the Horst Görtz Institute for IT-Security, Ruhr University Bochum, Bochum, Germany  
E-mail: {firstname.lastname}@rub.de.

**Our contribution:** Integrating a side-channel Trojan into an SCA-protected design is extremely challenging if the device will be evaluated by a third-party certification body. In this practical setting, the device should provide the desired SCA protection in 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. In many cases, our general Trojan concept does not require the addition of any logic (even a single gate) to the design, making it extremely hard to detect. In case of ASIC platforms, the Trojan may be introduced by slightly changing the characteristics of a few transistors, and for FPGA platforms by changing the routing of particular signals. Unlike previous work, it does not affect the provable-security feature of the underlying design unless the Trojan is triggered. Also, our technique is not based on the leakage of the PRNG. More precisely, our technique injects a *parametric* Trojan that can be triggered, 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 subverted device (or by decreasing its supply voltage) the Trojan is triggered and exhibits exploitable leakage. In order to avoid accidental triggering during regular operation and to make detection by evaluation labs unlikely, we choose a trigger frequency that is beyond the specified maximum operational frequency of the device (which is smaller than the actual maximum frequency of the design due to the inserted Trojan). As shown in the following sections, there is a carefully chosen gap between the specified maximum clock frequency of the device and the clock frequency where the Trojan is triggered. In other words, by increasing the clock frequency such that the critical path delay is violated, the device starts to operate faulty; by further increasing the clock frequency, the device operates again correctly while exhibiting SCA leakage, i.e., our inserted Trojan becomes active.

As a case study, we integrate this Trojan into a threshold implementation of the PRESENT cipher, which is a popular block cipher for embedded applications [11]. In contrast to the results presented in [50], we do not target FPGA platforms, but ASICs here. We succeeded in implementing the malicious design in two different low-power CMOS process technologies, 90 nm and 65 nm, as part of side-channel test chips. We present SCA evaluations of both ASICs based on real-silicon measurements, both when the Trojan is triggered and when it is not, which confirm the soundness of our approach. However, during the design process we identified a number of obstacles to overcome when integrating such a Trojan into lightweight ciphers implemented in advanced CMOS technology. As a result, these first Trojan implementations on ASIC platforms do not come at exactly zero overhead, but require the addition of a few cells (less than half a percent of the cipher circuit’s area). This result, however, does not contradict the potential stealthiness, in terms of zero overhead and negligible frequency reduction, of the general approach. In this regard, we detail in which cases such a Trojan can indeed be implemented at zero overhead and in which cases a few additional gates are needed.

**Outline:** Section II deals with necessary background and definitions in the areas of hardware Trojans and threshold implementations as an SCA countermeasure. Afterwards, in Section III we express our core idea how to insert our Trojan into a secure threshold implementation. In Section IV we give details on how to apply such a technique on a threshold implementation of the PRESENT cipher, and in Section V we explain in detail how to realize such an implementation in ASIC technology. Finally, in Section VI the corresponding results of an ASIC-based case study including SCA evaluations of the manufactured devices are presented.

## II. BACKGROUND

### A. Hardware Trojans

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. One 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 [12].

The goal of the Trojan can be achieved by a payload which can change the circuit functionally or leak its secret information. In [13] 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 [14], in which the actions can be categorized into classes of *modify functionality*, *modify specification*, *leak information*, and *denial of service*.

The work presented in [6] is concerned with 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. Another class of hardware Trojans – called Malicious Off-chip Leakage Enabled by Side-channels (MOLES) – has been presented in [1], 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. In [15] a design methodology for building stealthy parametric hardware Trojans and its application to Bug Attacks [16], [17] has been proposed. The Trojans are based on increasing the 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.

### B. Threshold Implementations

It can definitely be said that *masking* is the most-studied countermeasure against SCA attacks. It is based on the concept of *secret sharing*, where a secret  $x$  (e.g., intermediate values of a cipher execution) is represented by a couple of shares  $(x^1, \dots, x^n)$ . In case of an  $(n, n)$ -threshold secret sharing scheme, having access to  $t < n$  shares does not reveal any information about  $x$ . One of such schemes is Boolean secret sharing, also known as Boolean masking in the context of SCA, where  $x = \bigoplus_{i=1}^n x^i$ . Hence, if the entire computation of a cipher is conducted 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  $\mathcal{L}(\cdot)$  over the shares is straightforward since  $\mathcal{L}(x) = \bigoplus_{i=1}^n \mathcal{L}(x^i)$ . All the difficulties belong to implementing non-linear functions over such a shared representation. This concept has been applied in hardware implementations of AES (mainly with  $n = 2$ ) with no success [18]–[21] until the Threshold Implementation (TI) – based on sound mathematical foundations – has been introduced in [22], which defines the minimum number of 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  $y = \mathcal{F}(x)$  is desired, i.e.,  $(y^1, y^2, y^3) = \mathcal{F}^*(x^1, x^2, x^3)$ , where

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

Indeed, each output share  $y^{i \in \{1,2,3\}}$  is provided by a component function  $\mathcal{F}^i(\cdot, \cdot)$  which receives only two input shares. In other words, one input share is always 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  $(\mathcal{F}^1(x^2, x^3), \mathcal{F}^2(x^3, x^1), \mathcal{F}^3(x^1, x^2))$  form the shared output  $(y^1, y^2, y^3)$ .

In order to fulfill the above-given statement that “having access to  $t < n$  shares does not reveal any information about  $x$ ”, the shares need to follow a uniform distribution. For simplicity suppose that  $n = 2$ , and the shares  $(x^1, x^2)$  represent secret  $x$ . 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  $x^2 = x \oplus x^1$  and guess  $x$ . Hence, the security of masking schemes<sup>1</sup> relies on the uniformity of the masks. More precisely, when  $x^1 = m$ ,  $x^2 = x \oplus m$ , and  $m$  is taken from a randomness source (e.g., a PRNG), the distribution of  $m$  should be uniform (or let say with full entropy).

<sup>1</sup>Except those which are based on low-entropy masking [23], [24].

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  $\mathcal{F}^*(\cdot, \cdot, \cdot)$  as a TI of a bijective function  $\mathcal{F}(\cdot)$ , the *uniformity* property of TI is fulfilled if  $\mathcal{F}^*(\cdot, \cdot, \cdot)$  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 [22], [25]–[28]. Alternatively, the shares can be remasked at the end of every non-uniformly shared non-linear function (see [29], [30]), 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 [31], [32], 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 other cases including higher-order TI we refer the interested reader to the original articles [22], [25], [27].

### III. TECHNIQUE

As explained in the former section – by means of TI – it is possible to realize hardware cryptographic devices secure against certain SCA attacks. Our goal is to create a situation where 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 a higher clock frequency than the specified maximum operational frequency of the device, or a 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 [15] – we intentionally lengthen certain paths of a combinatorial circuit. This is done in such a way that – by increasing the device’s 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., 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 II-B – ignoring the uniformity – achieving a non-complete shared function  $\mathcal{F}^*(\cdot, \cdot, \cdot)$  of a given quadratic function  $\mathcal{F}(\cdot)$  is straightforward. Focusing on one output bit of  $\mathcal{F}(x)$ , and representing  $x$  by  $s$  input bits  $\langle x_s, \dots, x_1 \rangle$ , we can write

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

The coefficients  $k_0, \dots, k_{s-1, s} \in \{0, 1\}$  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* [25] as

- $\mathcal{F}_i^1(\cdot, \cdot)$  contains the linear terms  $x_i^2$  and the quadratic terms  $x_i^2 x_j^2$  and  $x_i^2 x_j^3$ .
- $\mathcal{F}_i^2(\cdot, \cdot)$  contains the linear terms  $x_i^3$  and the quadratic terms  $x_i^3 x_j^3$  and  $x_i^3 x_j^1$ .
- $\mathcal{F}_i^3(\cdot, \cdot)$  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  $\mathcal{F}(\cdot)$  and all three component functions  $\mathcal{F}^1(x^2, x^3)$ ,  $\mathcal{F}^2(x^3, x^1)$ ,  $\mathcal{F}^3(x^1, x^2)$  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  $\mathcal{F}(\cdot)$  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 [33] 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 [25] and for  $s = 5$  in [34]. Since the number of existing quadratic classes are restricted, a uniform TI can be found by exhaustive search. 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 [25]. Supposing that  $\mathcal{Q}^*(\cdot, \cdot, \cdot)$  is a uniform TI of  $\mathcal{Q}(\cdot)$ , 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  $\mathcal{F}(\cdot)$ :

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

This scenario has been followed in several works, e.g., [35]–[39].

- 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  $\mathcal{F}^1(x^2, x^3)$  and  $\mathcal{F}^3(x^1, x^2)$ . 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 [26], [40] to construct uniform TI of PRESENT and Keccak non-linear functions.

We should here refer to a similar approach called remasking [25], [30] where – instead of correction terms – fresh

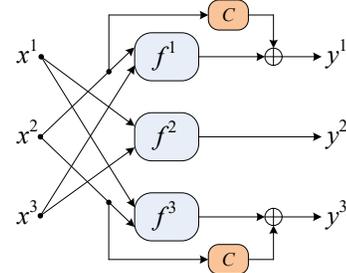


FIGURE 1: Exemplary TI construction with a correction term  $C$ .

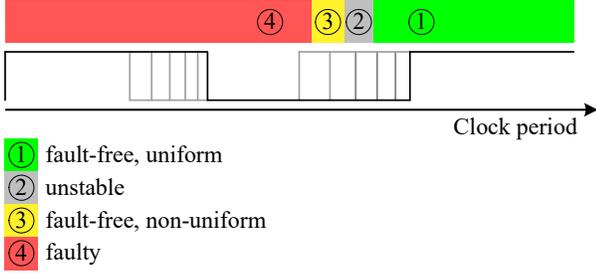
randomness is added to the output of the component functions to make the outputs uniform. In this case, obviously a certain number of fresh mask bits are required at every clock cycle (see [30], [41]).

Our technique is based on the second scheme explained above. If we make the paths related to the correction terms the longest, by increasing the clock frequency such paths are the first whose delays 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 ciphertexts assuming that the underlying design realizes an encryption function. It means that the design operates like an alternative design where no correction terms exist. 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 [42]), 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.

Based on Figure 2, which shows a corresponding timing diagram, the device’s 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



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

violated,

- by further 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 further 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 and ③ periods, which are of crucial importance. If they are 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 required in period ③, to make sure that the correct sensitive intermediate values are processed by the circuit and revealed to the Trojan attacker through their side-channel leakage.

#### IV. APPLICATION

In order to show an application of our technique, we focus on a first-order TI design of the PRESENT cipher [11] as a case study. The PRESENT Sbox is a 4-bit cubic bijection  $\mathcal{S} : \text{C56B90AD3EF84712}$ . 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 an extra register between  $\mathcal{F}^*$  and  $\mathcal{G}^*$  (i.e., TI of  $\mathcal{F}$  and  $\mathcal{G}$ ). As shown in [25],  $\mathcal{S}$  is affine equivalent to class  $\mathcal{C}_{266} : \text{0123468A5BCFED97}$ , which can be decomposed to quadratic bijections with uniform TI. The works reported in [36], [37], [43] 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 III).

However, the first TI of PRESENT has been introduced in [26], where the authors have decomposed the Sbox by  $\mathcal{G} : \text{7E92B04D5CA1836F}$  and  $\mathcal{F} : \text{08B7A31C46F9ED52}$ . They have accordingly provided a 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. Contrary to all other related works, we first try to find a **non-uniform** TI of

$\mathcal{G}(\cdot)$ , and we later make it uniform by means of correction terms. We start with the ANF of  $\mathcal{G}(\langle d, c, b, a \rangle) = \langle g_3, g_2, g_1, g_0 \rangle$ :

$$\begin{aligned} g_0 &= 1 \oplus a \oplus dc \oplus db \oplus cb, & g_2 &= 1 \oplus c \oplus b, \\ g_1 &= 1 \oplus d \oplus b \oplus ca \oplus ba, & g_3 &= c \oplus b \oplus a. \end{aligned}$$

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

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

$$\begin{aligned} y_0^2 &= a^3 \oplus d^3 c^3 \oplus d^1 c^3 \oplus d^3 c^1 \oplus d^3 b^3 \oplus d^1 b^3 \oplus d^3 b^1 \oplus \\ &\quad c^3 b^3 \oplus c^1 b^3 \oplus c^3 b^1 \end{aligned}$$

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

$$\begin{aligned} y_0^3 &= a^1 \oplus d^1 c^1 \oplus d^1 c^2 \oplus d^2 c^1 \oplus d^1 b^1 \oplus d^1 b^2 \oplus d^2 b^1 \oplus \\ &\quad c^1 b^1 \oplus c^1 b^2 \oplus c^2 b^1 \end{aligned}$$

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

with  $\mathbf{x}^{i \in \{1,2,3\}} = \langle d^i, c^i, b^i, a^i \rangle$ . This is not a uniform sharing of  $\mathcal{G}(\cdot)$ , and by searching through possible correction terms we found three correction terms  $c^1 b^1$ ,  $c^2 b^2$ , and  $c^3 b^3$  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

$$\begin{aligned} \mathcal{C}^1(\mathbf{x}^2, \mathbf{x}^3) &= c^2 b^2 \oplus c^3 b^3, \\ \mathcal{C}^2(\mathbf{x}^3, \mathbf{x}^1) &= c^1 b^1 \oplus c^3 b^3, \\ \mathcal{C}^3(\mathbf{x}^1, \mathbf{x}^2) &= c^1 b^1 \oplus c^2 b^2, \end{aligned}$$

and adding them respectively to  $y_1^1$ ,  $y_2^1$ , and  $y_3^1$ , 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  $c^2 b^2$  which should be added to  $\mathcal{G}^1(\cdot, \cdot)$  and  $\mathcal{G}^3(\cdot, \cdot)$ .

##### A. Inserting the Trojan

We explain how to realize the Trojan functionality by path delay fault model [44], without modifying the logic circuit. The Trojan can be triggered by violating the delay of the combinatorial logic paths that pass through the targeted correction terms  $c^2 b^2$ . 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 delays increase and add up to the path delay faults.

Given in [15], 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<sup>2</sup>. 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.

1) *ASIC Platforms*: In an ASIC platform, the necessary delays for such Trojans can be introduced in a multitude of ways. Apart from the addition of extra gates to the chosen paths there is also the attractive option to achieve the same goal by slight modifications on the sub-transistor level so that only the parameters of a few transistors of the design are changed. To increase the delays of transistors in stealthy ways, there are many possible ways in practice. Such a Trojan is very difficult to be detected by e.g., functional testing, visual inspection, or 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 be present at higher abstraction levels and hence cannot be detected at those levels, because the actual Trojan is inserted at the sub-transistor level. There are several stealthy ways to slightly change the parameters of transistors of a gate and make it slower. Exemplarily, we list three methods below.

**Decreasing 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 an extremely narrow width, if the lower level information of the gate is available in the library (e.g., SPICE model).

**Raising the Threshold**: A common way of increasing the 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-V<sub>t</sub> design) is very common approach and provides the designer with 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.

**Increasing the Gate Length**: Gate length biasing can increase delay of a gate by reducing the current of its transistors [45].

2) *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 using *Through Switch Boxes* to make certain paths longer. 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 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, and hence is hard to detect particularly if the utilization reports are compared.

Focusing on our target, the only paths which should be lengthened are both instances of  $c^2b^2$  in  $\mathcal{G}^1(.,.)$  and  $\mathcal{G}^3(.,.)$ . 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  $\mathcal{G}^3(.,.)$ . In the following section we explain in detail which algorithms, metrics and heuristics need to be used to select and lengthen the correct paths in arbitrary ASIC designs of such kind.

## V. ASIC IMPLEMENTATION

For ASIC platforms, we utilize the stealthy parametric Trojan introduced in [15]. It consists of two main phases: *path selection phase* and *delay distribution phase*. We briefly explain each of these phases in subsections V-A and V-B. Our goal is to make the paths related to our target correction term, which is added to two component functions, the longest so that by increasing the clock frequency such paths are the first whose delays are violated. The paths must be very carefully selected and 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.

### A. Rare Path Selection Phase

The path selection phase seeks to find a path  $\pi$  through the netlist of the circuit that passes through the targeted correction term. Note that the delays are not considered in this phase of the work. Path  $\pi$  is initialized to contain a transition on the targeted correction term node. This initial single-node path

<sup>2</sup>It means that the selected paths are the only ones in the circuit whose critical delay can be violated.

**Require:** A single node  $\pi$  in the netlist of the circuit  
**Ensure:** A sensitizable path  $\pi$  starting at a primary input and ending at a primary output

```

1: while ( $\pi$  does not start at a primary input) do
2:   new_node_candidates = {All transitions that can be prepended to  $\pi$ }
3:   Order new_node_candidates by difficulty of justification.
4:   for (each member  $n'$  of new_node_candidates) do
5:     new_subpath  $\pi' =$  prepend  $n'$  to the tail of  $\pi$ 
6:     if (check-SAT( $\pi'$ )) then
7:        $\pi = \pi'$ 
8:       Exit for loop.
9:     end if
10:  end for
11: end while
12: while ( $\pi$  does not end at a primary output) do
13:   new_node_candidates = {ALL transitions that can be appended to  $\pi$ }
14:   Order new_node_candidates by difficulty of propagation.
15:   for (each member  $n'$  of new_node_candidates) do
16:     new_subpath  $\pi' =$  append  $n'$  to the head of  $\pi$ 
17:     if (check-SAT( $\pi'$ )) then
18:        $\pi = \pi'$ 
19:       Exit for loop.
20:     end if
21:   end for
22: end while

```

**ALGORITHM 1:** Extracting a hard to trigger sensitizable path passing through a specific node.

$\pi$  is then extended incrementally backwards until reaching the primary inputs, and extended incrementally forward until reaching the primary outputs. The path selection algorithm is given in Alg. 1. Starting from the first transition on the current path  $\pi$ , we repeatedly try to extend the path back towards the PIs by prepending one new transition to the path. To select such a transition, the algorithm creates a list of candidate transitions that can be prepended to the path, which is sorted according to the difficulty of creating the necessary conditions to justify the transition. Whenever a node is prepended to  $\pi$  to create a candidate path  $\pi'$ , the sensitizability of  $\pi'$  is checked by calling *check-SAT* function. In this function SAT-based techniques [46] are used to check sensitizability of the path. If the SAT solver returns SAT, then path  $\pi'$  is known to be a subpath of a sensitizable path from a primary input to a primary output. If this newly added tail node is not a primary input, then the algorithm will again try to extend it backwards.

The forward propagation part is similar to the aforementioned backward propagation, except that it adds nodes to the head of the path until reaching a primary output. At each step of the algorithm, a list of candidates is again formed. In this case, they are ordered according to difficulty of propagation instead of difficulty of justification. Each time a new candidate path is created by adding a candidate node to the existing path, a SAT check is again performed to ensure that the nodes are only added to  $\pi$  if it remains sensitizable.

## B. Delay Distribution Phase

Once paths are selected, the delay of them must be increased so that the total path delays exceed the clock period and errors occur when the paths are sensitized. Choosing where to add delay on the paths must be done carefully, because the gates along the chosen paths are also part of many other intersecting or overlapping paths. Any delay added to the chosen paths therefore may cause errors even when the chosen paths are

not sensitized. Genetic algorithm is used to smartly decide the delay of each gate along with some constraints to restrict the allowed solution space, and a fitness function for evaluating solutions.

**Total Path Delay Constraint:** Assume each of the chosen paths  $\pi$  includes  $n$  gates and target path delay is  $D$ . This constraint specifies that the sum of assigned delays along the path is equal to the target path delay  $D$ . To cause an error,  $D$  must exceed the period  $\Phi$ .

$$D = \sum_{i=0}^n d_i \quad (1)$$

**Gate Delay Constraint:** Assume  $d'_i$  represents the nominal delay of the  $i^{th}$  gate on the chosen path  $\pi$ , and  $s_i$  represents the slack metric associated with the same gate. Each slack parameter  $s_i$  describes how much delay can be added to the corresponding gate without causing the path to exceed the period  $\Phi$ . The slack for each gate is computed as a function of the nominal delay of the gate, data dependency, and the clock period [48], [49]. Because the targeted path delay  $D$  does exceed the period  $\Phi$ , gate delays are allowed to exceed their computed slack. The following equation shows this constraint where  $c$  is a constant.

$$d'_i + s_i - c \leq d_i \leq d'_i + s_i + c \quad (2)$$

**Fitness Function:** The cost function consists of two parts; *i*) the faults cancel each others' effect, i.e., the faults on targeted correction term in two functions  $\mathcal{G}^1$  and  $\mathcal{G}^3$  happen at the same time and cancel the effect of each other so that the functionality of the design is not altered, and *ii*) the design does not fulfill the uniformity property anymore. To cover both cases in our final cost function we define it as the following equation in where the first term corresponds to case (i) and the second term corresponds to case (ii). Our goal is to minimize this cost function.

$$\text{Cost}_F(d_1, \dots, d_n) = \text{ErrorRate}_{\text{design}} + 1/\text{ErrorRate}_{\mathcal{G}^1 \text{ and } \mathcal{G}^3} \quad (3)$$

We use random simulation to evaluate the cost of any delay assignment. When the genetic algorithm in Matlab [47] needs to evaluate the cost of a particular delay assignment, it does so by executing a timing simulator. The timing simulator, in our case ModelSim, applies test vectors to the circuit-under-evaluation and a golden copy of the circuit and compares the respective outputs to count the number of errors.

## VI. ASIC PRACTICAL RESULTS

In this section we describe how we have designed and implemented first ASIC prototypes incorporating such a malicious design. We then verify that the resulting chips are indeed resistant against side-channel attacks when the Trojan is not triggered and that this resistance can be nullified when triggering it.

Section ?? demonstrated by practical experiments that the proposed hardware Trojan and the presented implementation techniques are valid on FPGA-based platforms. Here, we aim to provide a similar case study, but with respect to

ASIC platforms. In this regard we carried out the described design stages and implemented the trojanized PRESENT threshold implementation circuit in two different process technologies, 90 nm and 65 nm low power CMOS. Both ASICs, which can be seen in Figure 3, were developed using an identical design procedure, including the usage of low, high and standard threshold voltage cells, and were manufactured by the same foundry.

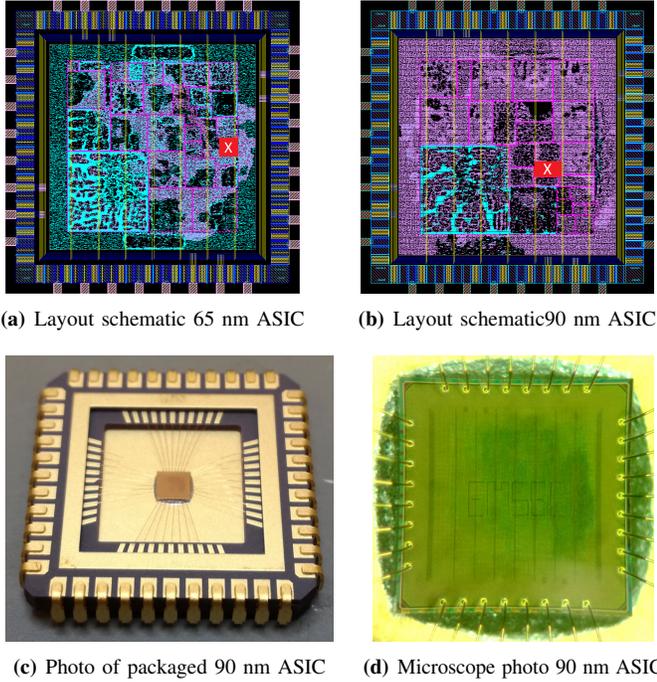


FIGURE 3: Layout schematic and photos of the ASIC prototypes.

The size of both chips is 2mm × 2mm. The side-channel resistant PRESENT TI cores containing the parametric SCA Trojans have been placed and routed in clearly delimited rectangular areas, which are marked in red color with a white cross in both layout schematics 3(a) and 3(b), taken from the *Synopsys IC Compiler (Version 2016.12)* software.

We made use of the malicious PRESENT TI S-Box that has been introduced in the previous sections and embedded it in a design with full encryption functionality shown in Figure ??.

Synthesizing the unaltered threshold implementation of the PRESENT S-Box (i.e., without the inserted Trojan) in the 90 nm and 65 nm target libraries revealed that the design could potentially meet clock frequencies in the GHz range, even when operated under worst case operating conditions (i.e., low supply voltage, high temperature). Unfortunately, no digital IO cells were available in our target technologies that could reliably propagate such a high-frequency clock into the circuit. Thus, when inserting the Trojan in the proposed way, i.e., by subtle manipulations at the sub-transistor level, and keeping period ③ small and stealthy, it could never be triggered, due to the restrictions of the IO cells and the extremely high performance of the circuit in the target technologies.

This observation already shows that implementing and testing such a design on an ASIC is more challenging than on an FPGA, due to the much higher performance of ASICs. In this regard

we have to conclude that an ultra-lightweight block cipher implementation like the serialized PRESENT, implemented in an advanced CMOS technology with small propagation delays, may not be the optimal choice for integrating such a Trojan on an ASIC in the most stealthy way. Yet, to keep the results comparable to those in [50], we stick to this example and find a workaround for the IO restriction.

Another difficulty when developing ASIC prototypes is the extensive amount of time and monetary resources that have to be invested. Thus, it is desirable to obtain a fully functioning prototype in the first attempt when designing a test chip. However, this is particularly difficult to achieve when the functionality of the design depends highly on the exact timing of certain signal paths in such a way that even small deviations from the predicted behavior can invalidate crucial assumptions. In such a case the designer has to trust its foundry that the characterized timing information included in the standard cell libraries and simulation models perfectly reflects the reality – which is hardly ever possible due to process variations. Thus, even commercial IC design houses often require multiple generations of prototypes that need to be characterized and adapted between each iteration to finally end up with a marketable end-user product. Unlike FPGA platforms where a new HDL design can be synthesized and implemented within a few minutes and without any additional cost, which allows for trial & error approaches, an IC implementation requires at least several months per tape out as well as a significant amount of money, even when sharing a wafer between multiple projects. Thus, for our case study, in order to not require multiple IC manufacturing iterations, but rather obtain a working prototype in the first attempt, we chose to limit the potential sources of error at the cost of sacrificing a part of the potential stealthiness of the Trojan. In particular, we chose to realize the delay which needs to be distributed among the selected paths partially by so-called delay gates<sup>3</sup> and optimize for a broad frequency range that triggers the Trojan while the PRESENT core still encrypts correctly (i.e., period ③). A delay gate does not have any logical functionality but simply propagates its input signal with a certain propagation delay to its output. Clearly, inserting delay gates into the masked S-Box makes the Trojan less stealthy than sub-transistor level modifications. The same is true for a significant reduction of the overall operating frequency of the circuit as it can be observed in the results presented in the following (this reduction is necessary due to the restrictions of the IO cells). However, we would like to stress that this case study is simply proving the conceptual soundness of the approach, in the sense that inserting this delay-based Trojan makes a side-channel resistant implementation vulnerable when increasing the clock frequency beyond a certain point. It is planned to demonstrate the stealthiness of the Trojan on ASIC platforms in further case studies. In many cases, for example targeting more complex non-linear functions (like the AES S-Box) or less advanced CMOS technologies (implying larger delays), such a use of

<sup>3</sup>Those gates were required since selecting even the slowest cells (high threshold voltage, low drive strength) could not add enough delay in order to make the Trojan triggerable through the IO cells.

| Technology node | Area w/o Trojan | Area w/ Trojan | Overhead |
|-----------------|-----------------|----------------|----------|
| 65 nm           | 4988.5 GE       | 5006.5 GE      | +0.36%   |
| 90 nm           | 4807.8 GE       | 4825.8 GE      | +0.37%   |

**TABLE I:** Area comparison (post-layout) of PRESENT TI implementation with and without inserted Trojan (realized by delay gates).

| Status | 65 nm ASIC                            | 90 nm ASIC                            |
|--------|---------------------------------------|---------------------------------------|
| ①      | $f \leq 33$ MHz                       | $f \leq 56$ MHz                       |
| ②      | $33 \text{ MHz} < f \leq 38$ MHz      | $56 \text{ MHz} < f \leq 61$ MHz      |
| ③      | $38 \text{ MHz} < f \leq$ appr. 1 GHz | $61 \text{ MHz} < f \leq$ appr. 1 GHz |
| ④      | appr. 1 GHz $< f$                     | appr. 1 GHz $< f$                     |

**TABLE II:** Frequency ranges for the different design states.

additional delay gates will not be required since the critical path of the design actually restricts the maximum operating frequency of the design (and not the limitations of the IO cells). Again, we chose the PRESENT threshold implementation as a case study here to keep the results comparable to [50]. And even in our case, where we particularly aimed for a broad range of period ③, the overhead in terms of area is very small, even less than half a percent as apparent from Table I. The range of clock frequencies that cause a certain state of the trojanized design can be seen in Table II. As described before, state ③ has the broadest frequency range and can easily be targeted by setting the clock frequency above 38 MHz for the 65 nm ASIC and 56 MHz for the 90 nm ASIC. The upper limit where the output of the circuit becomes faulty is an approximation, since it could not be determined experimentally due to the limitation of the IO cells.

#### A. Measurement Setup

In order to perform the SCA evaluations on the ASIC prototypes we built a simple custom measurement board. Since the ASICs have been packaged in JLCC-44 package (see Figure 3(c)), the custom board provides a corresponding PLCC-44 socket as well as connectors for a BASYS-3 FPGA board (containing an Artix-7 FPGA) to control the communication between PC and the ASIC. We measured the power consumption of the ASICs in the  $V_{dd}$  path by means of a digital sampling oscilloscope at a fixed sampling rate of 200 samples per clock cycle. Since the operating frequency varies between the different scenarios (Trojan triggered or not triggered), fixing the number of samples per clock cycle (instead of per time period) is the most fair evaluation method.

#### B. SCA Results

We evaluate the SCA resistance of our designs in three different settings using a non-specific t-test (fixed versus random) [51], [52] to examine the existence of detectable leakage. First, to validate the correct functionality of 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. Afterwards, we activate the PRNG and operate the design at low frequency in order to not activate the Trojan. Then, when the PRNG is still running we increase the clock frequency in order to activate the Trojan. In the latter case we also conduct key-recovery attacks.

1) *Results on 90 nm ASIC:* We first collected 1,000,000 traces with PRNG switched off when the design is operated at 25 MHz, i.e., the Trojan is not triggered. We followed the concept given in [52] for the collection of traces belonging to fixed and random inputs. Figure 4 shows the corresponding t-test results.

As expected a significant amount of detectable leakage can be observed in all moments, confirming the validity of the setup and the developed evaluation tools.

To repeat the same process when the PRNG is turned on, i.e., the masks for initial sharing of the plaintext are randomly chosen and uniformly distributed, we collected 50,000,000 traces for non-specific t-test evaluations. In this case, the device still operates at 25 MHz, i.e., the Trojan is not triggered. The corresponding results are shown in Figure 5.

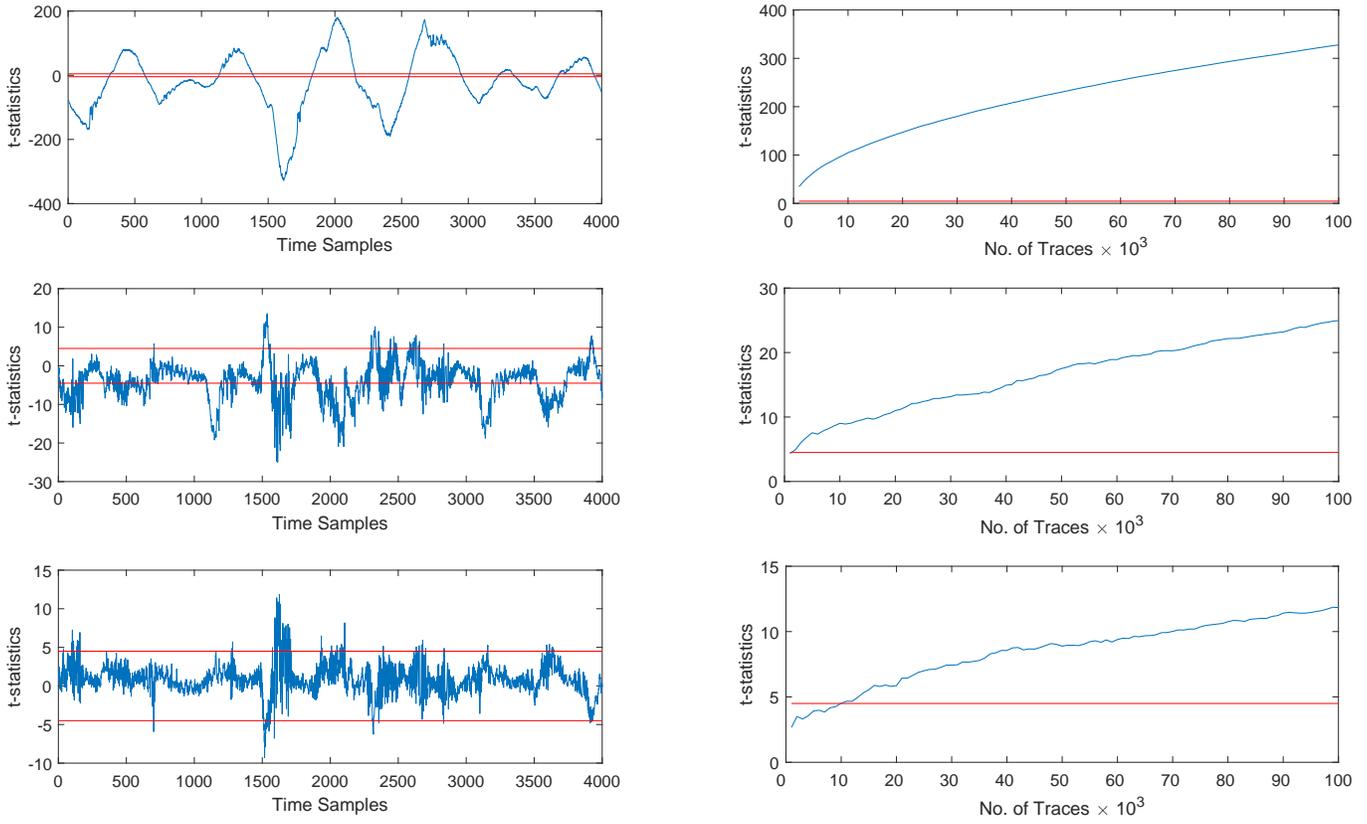
It can be seen that no leakage is detected in any of the three statistical moments after 50,000,000 traces. However, when observing the progress of the maximum absolute t-value in the second-order moment over the number of traces one may notice that the 4.5 threshold is occasionally exceeded. We should emphasize here that the underlying TI construction is a first-order masking, which can provide provable security against first-order SCA attacks. However, higher-order attacks (in this case second-order attacks already) are expected to exploit the leakage, but they are sensitive to the noise level [53] since accurately estimating higher-order statistical moments requires huge amounts of samples compared to lower-order moments. Thus, the second-order leakage is not unexpected, but the noise level seems too large to reliably detect (or exploit) this leakage.

As the last step, the same scenario is repeated when the clock frequency is increased to 85 MHz, where the design is in the ③ period, i.e., with correct functionality and without uniformity. Similar to the previous experiment, we collected 50,000,000 traces for a non-specific t-test, whose results are shown in Figure 6.

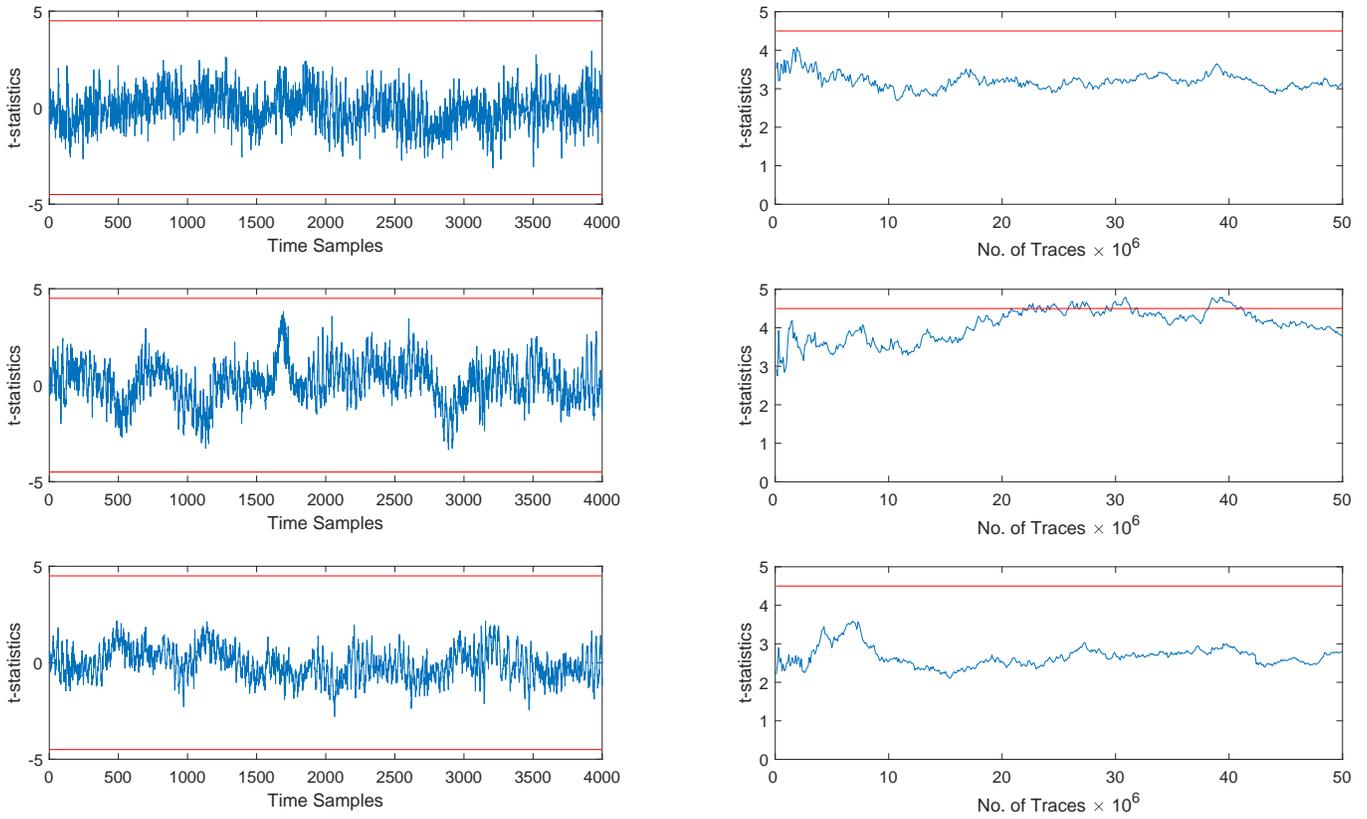
As shown by the graphics, there is detectable leakage through the first and second statistical moment but with lower t-statistics compared to the case with PRNG off. Therefore, we also have to examine the feasibility of key recovery attacks. To this end, we made use of those collected traces which are associated with random inputs, i.e., around 25,000,000 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 attack [4] targeting a key nibble by predicting an S-Box output bit at the first round of the encryption. As an example, Figure 7 presents an exemplary corresponding result.

2) *Results on 65 nm ASIC:* After we have seen that the Trojan indeed achieves what it has been designed for on the 90 nm ASIC, we repeat the same kind of experiments on the 65 nm chip. At first, the results after 1,000,000 traces with the deactivated Trojan (25 MHz) and the switched off PRNG can be seen in Figure 8.

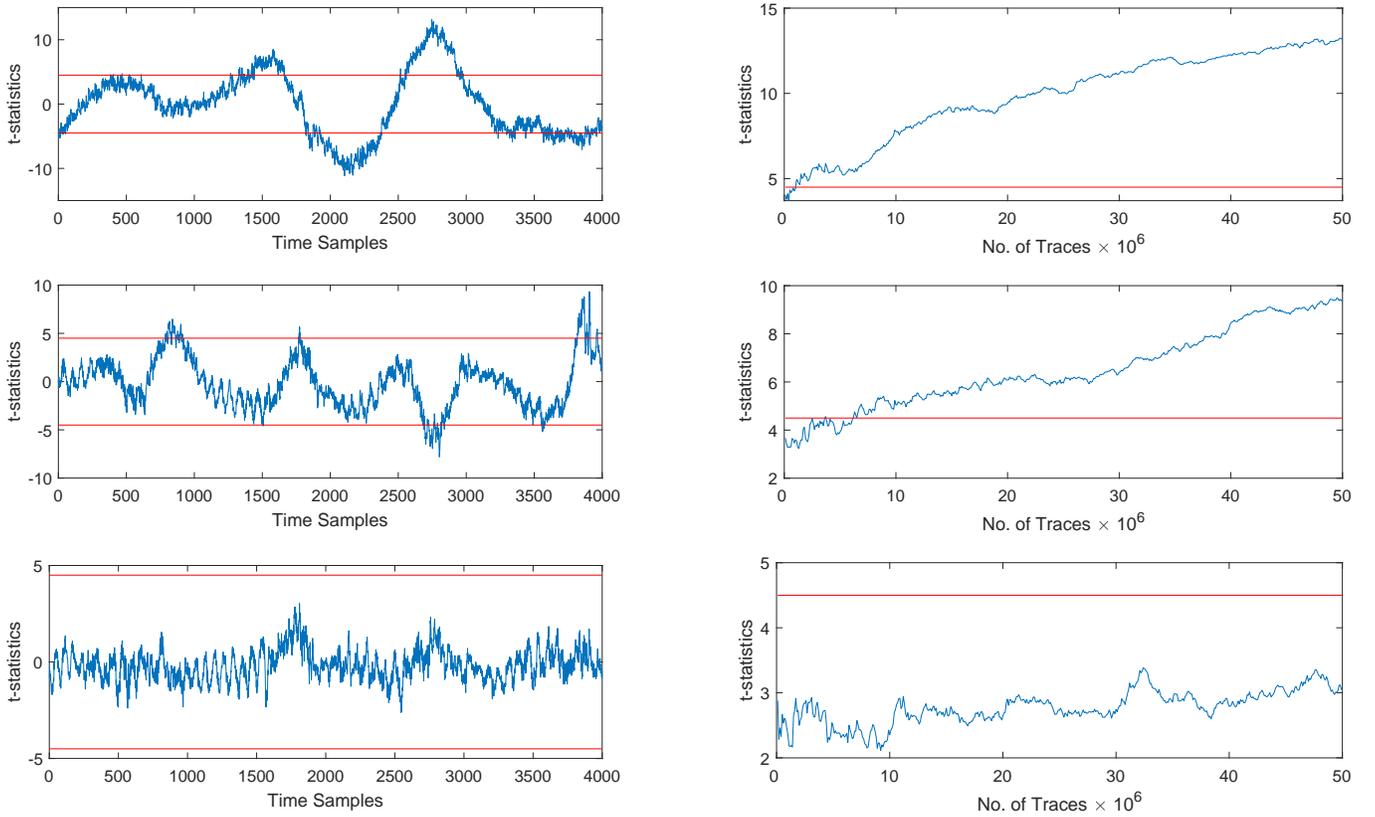
As before, detectable leakage is visible in all three statistical moments, but its magnitude is significantly smaller than on the 90 nm ASIC, indicating a lower signal-to-noise ratio. Thus, for the next step with PRNG on we measured more traces than



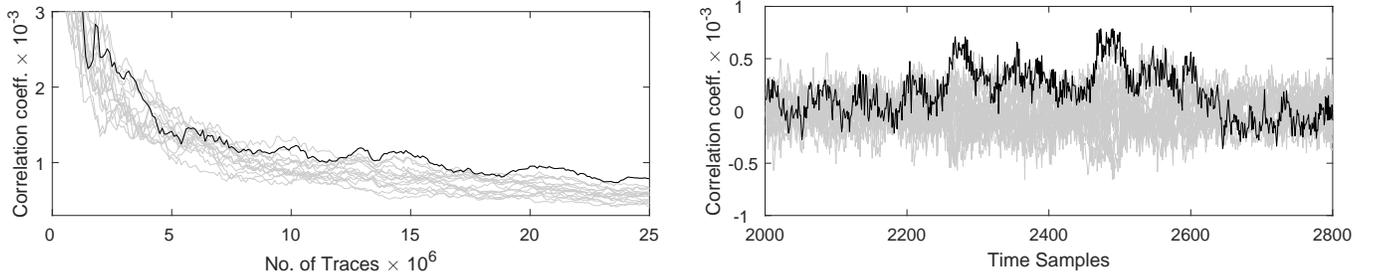
**FIGURE 4:** 90 nm ASIC, PRNG off, clock frequency 25 MHz (trojan not triggered),  $t$ -test results with 1 million traces (left), absolute maximum  $t$ -value over the number of traces (right).



**FIGURE 5:** 90 nm ASIC, PRNG on, clock frequency 25 MHz (trojan not triggered),  $t$ -test results with 50 million traces (left), absolute maximum  $t$ -value over the number of traces (right).



**FIGURE 6:** 90 nm ASIC, PRNG on, clock frequency 85 MHz (trojan triggered),  $t$ -test results with 50 million traces (left), absolute maximum  $t$ -value over the number of traces (right).



**FIGURE 7:** 90 nm ASIC, PRNG on, clock frequency 85 MHz (trojan triggered), CPA results targeting a key nibble based on an S-Box output bit with 25 million traces (right), absolute maximum correlation coefficient over the number of traces (left).

before, namely 80,000,000. The results in Figure 9 show that with PRNG on and the Trojan not triggered at 25 MHz clock, there is no detectable leakage in any moment.

When measuring at 50 MHz, however, i.e., triggering the Trojan, significant leakage can be detected in all moments, as apparent in Figure 10.

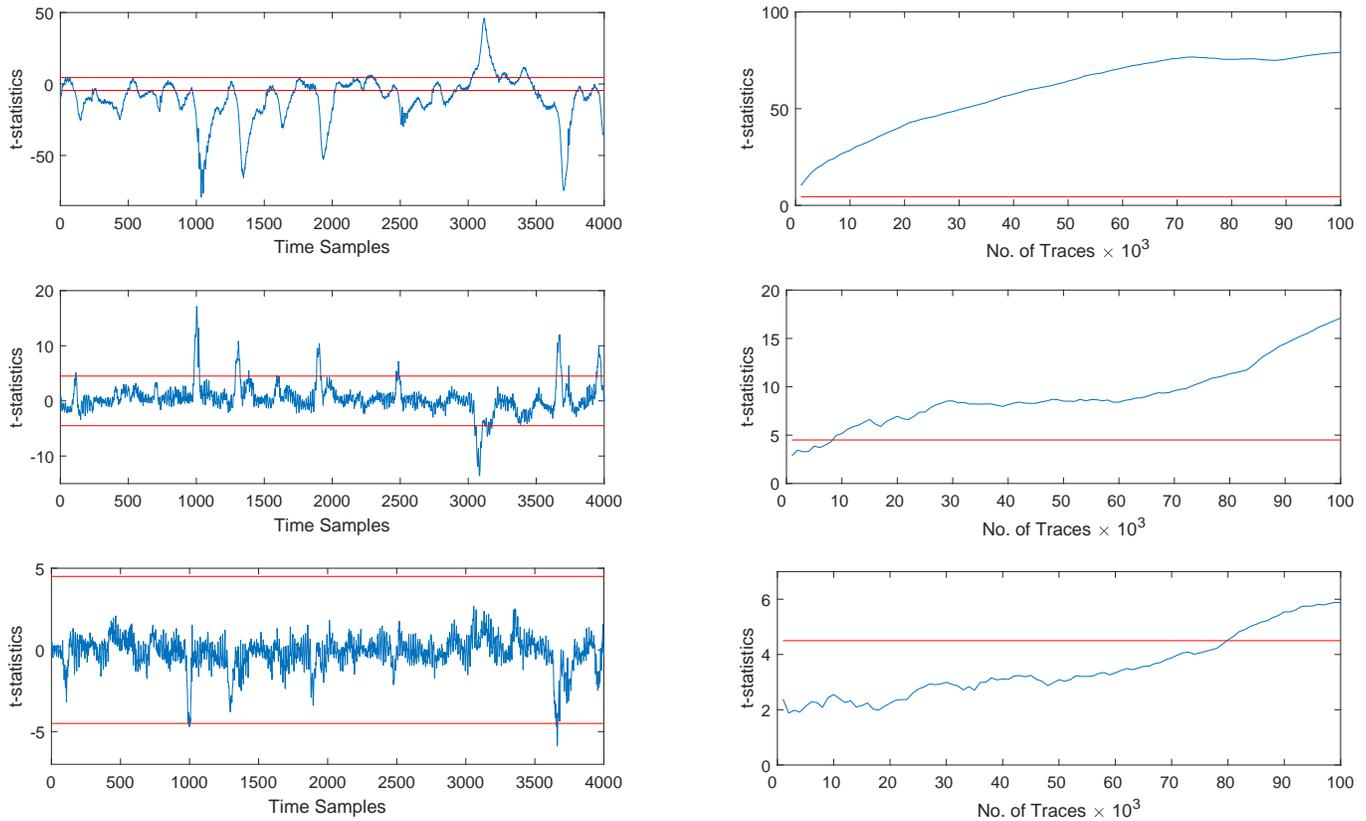
The successful CPA in 11 targeting a key nibble based on an S-Box output bit using 40,000,000 traces confirms that the leakage is indeed exploitable.

## VII. CONCLUSIONS

We show how a parametric hardware Trojan with very low overhead can be inserted into SCA-resistant designs. The presented Trojan is capable of being integrated into both ASIC and FPGA platforms. This kind of parametric Trojan is very hard to be detected, since it bears the potential to be inserted without the addition or removal of any logic into or from the

target design. Thus, even in a white-box scenario the Trojan remains stealthy and is unlikely to be detected by an evaluation lab.

The underlying concept is to lengthen certain paths of a combinatorial logic realizing a non-linear function under the foundations of threshold implementation, in such a way that by violating their delay (i.e., by running the device at a high frequency) the uniformity property of the utilized masking scheme is not fulfilled anymore. Our case study based on two ASIC prototypes, while admittedly suffering from some shortcomings, shows clearly that increasing the clock frequency triggers the malicious threshold implementation design to start leaking exploitable information through side channels. Hence, the Trojan adversary can activate the Trojan and make use of the exploitable leakage, while the design can pass SCA evaluations when the Trojan is not triggered. To the best of our knowledge, compared to the previous works in the areas



**FIGURE 8:** 65 nm ASIC, PRNG off, clock frequency 25 MHz (Trojan not triggered),  $t$ -test results with 1 million traces (left), absolute maximum  $t$ -value over the number of traces (right).

of side-channel hardware Trojans, our construction is the only one which is applied on a first-order provably-secure SCA countermeasure, and is parametric with very low (or even no) overhead.

In fact, by decreasing the supply voltage the same effect can be seen. 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 [54], where the difficulties of embedding a “clock frequency monitor” in presence of supply voltage changes are shown.

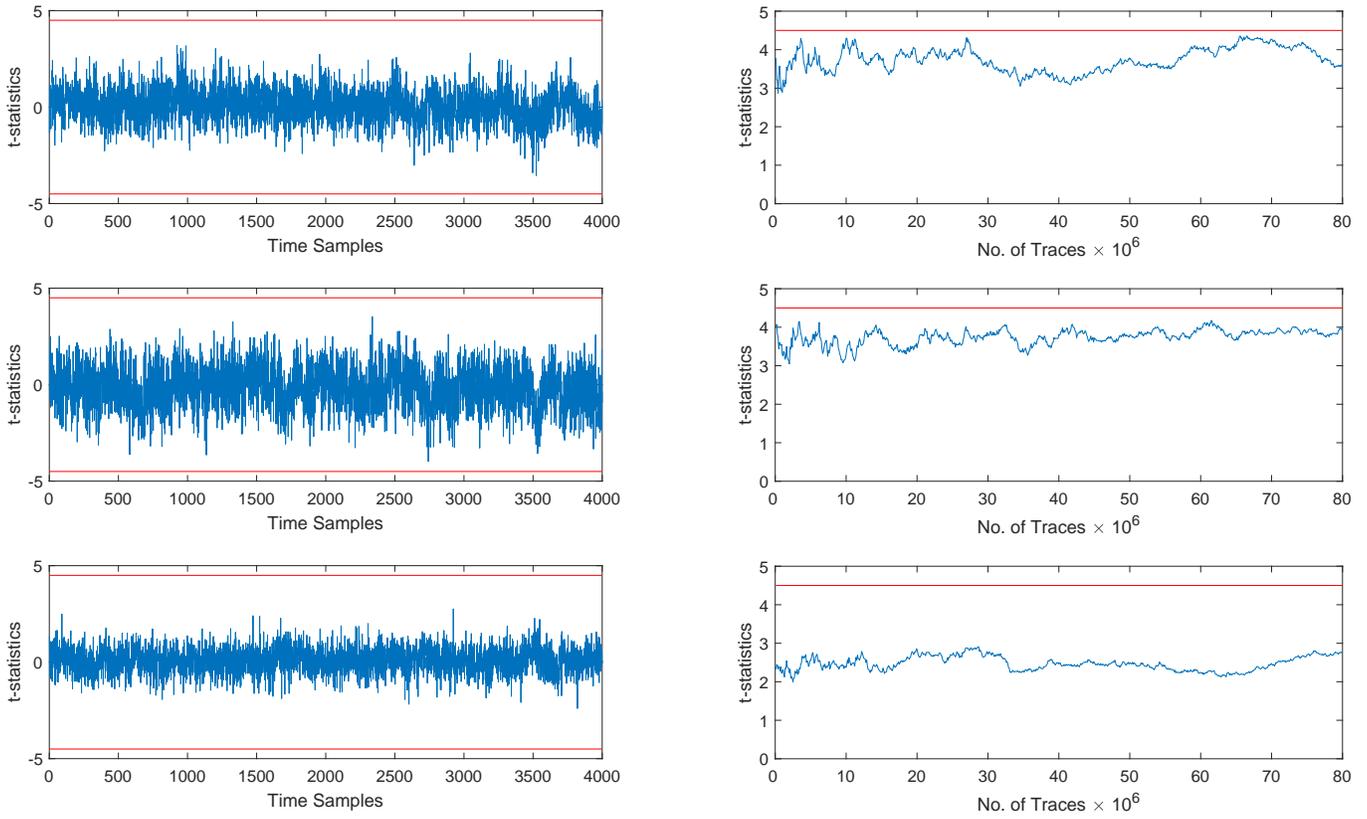
#### ACKNOWLEDGMENTS

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 271752544 “NaSCA: Nano-Scale Side-Channel Analysis”.

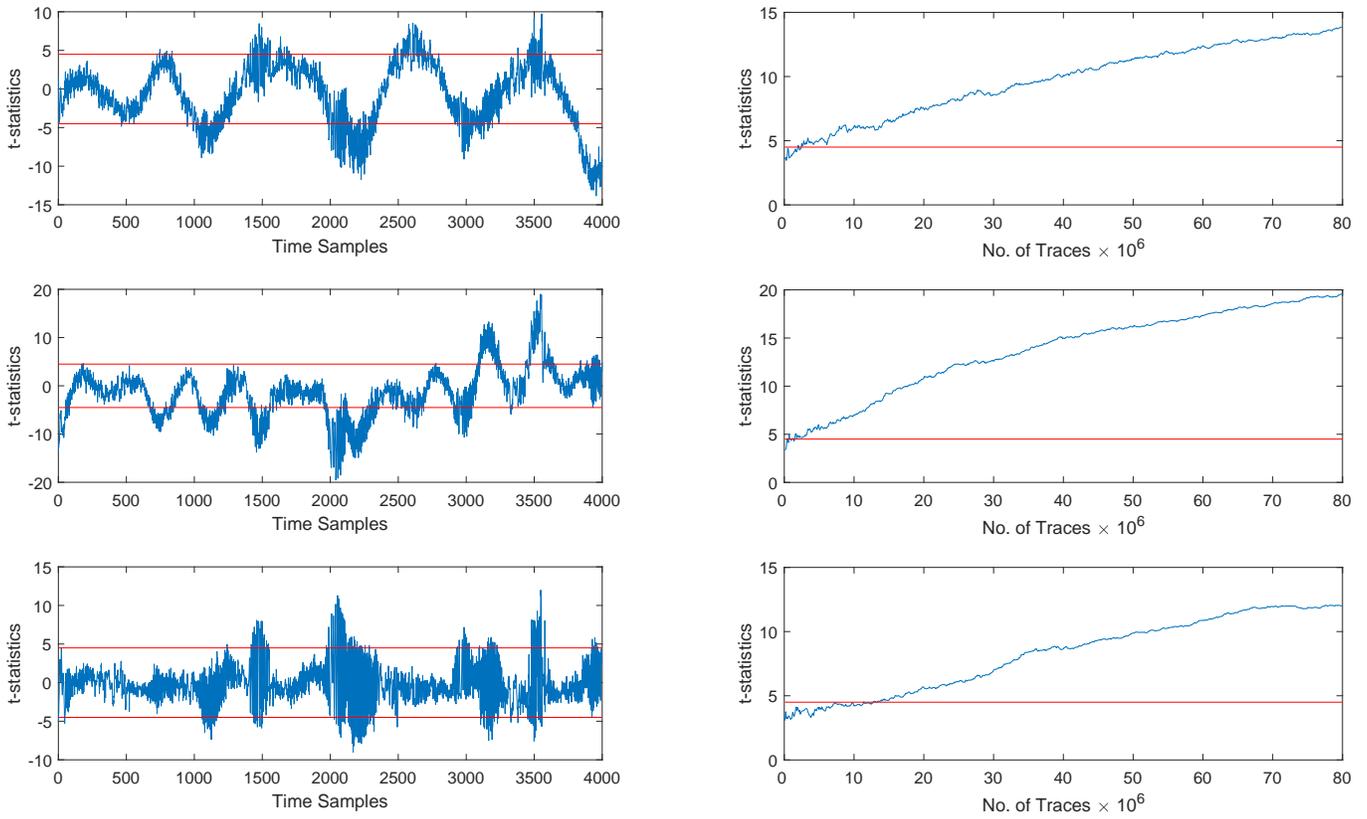
#### REFERENCES

- [1] L. Lin, W. Burleson, and C. Paar, “MOLES: Malicious off-chip leakage enabled by side-channels,” in *ICCAD 2009*. ACM, 2009, pp. 117–122.
- [2] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, “Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering,” in *CHES 2009*, ser. Lecture Notes in Computer Science, vol. 5747. Springer, 2009, pp. 382–395.
- [3] P. C. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” in *CRYPTO 1996*, ser. Lecture Notes in Computer Science, vol. 1109. Springer, 1996, pp. 104–113.
- [4] P. C. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *CRYPTO 1999*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397.

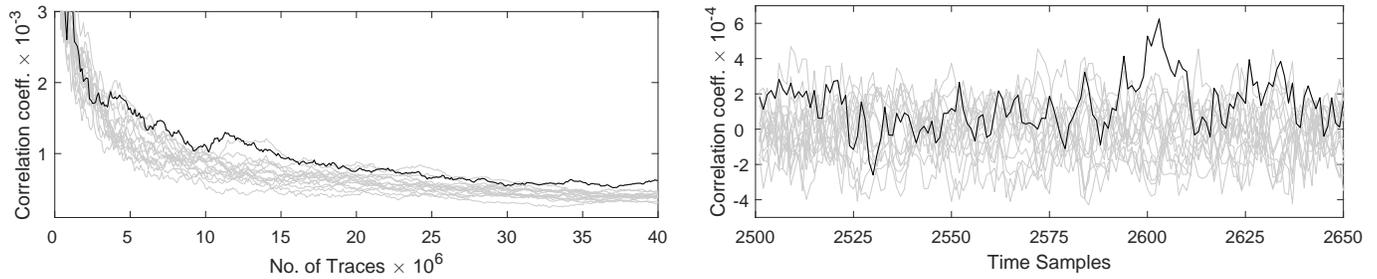
- [5] M. Kasper, A. Moradi, G. T. Becker, O. Mischke, T. Güneysu, C. Paar, and W. Burleson, “Side channels as building blocks,” *J. Cryptographic Engineering*, vol. 2, no. 3, pp. 143–159, 2012.
- [6] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, “Stealthy Dopant-Level Hardware Trojans,” in *CHES 2013*, ser. Lecture Notes in Computer Science, vol. 8086. Springer, 2013, pp. 197–214.
- [7] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard, “Evaluation of the Masked Logic Style MDPL on a Prototype Chip,” in *CHES 2007*, ser. Lecture Notes in Computer Science, vol. 4727. Springer, 2007, pp. 81–94.
- [8] A. Moradi, M. Kirschbaum, T. Eisenbarth, and C. Paar, “Masked Dual-Rail Precharge Logic Encounters State-of-the-Art Power Analysis Methods,” *IEEE Trans. VLSI Syst.*, vol. 20, no. 9, pp. 1578–1589, 2012.
- [9] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, “A2: Analog malicious hardware,” in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 18–37.
- [10] Y. Hou, H. He, K. Shamsi, Y. Jin, D. Wu, and H. Wu, “R2d2: Runtime reassurance and detection of a2 trojan,” in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2018, pp. 195–200.
- [11] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An Ultra-Lightweight Block Cipher,” in *CHES 2007*, ser. Lecture Notes in Computer Science, vol. 4727. Springer, 2007, pp. 450–466.
- [12] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, “Hardware Trojan: Threats and emerging solutions,” in *HLDVT 2009*. IEEE Computer Society, 2009, pp. 166–171.
- [13] Y. Jin and Y. Makris, “Hardware Trojan Detection Using Path Delay Fingerprint,” in *HOST 2008*. IEEE Computer Society, 2008, pp. 51–57.
- [14] X. Wang, H. Salmani, M. Tehranipoor, and J. F. Plusquellic, “Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis,” in *DFT 2008*. IEEE Computer Society, 2008, pp. 87–95.
- [15] S. Ghandali, G. T. Becker, D. Holcomb, and C. Paar, “A Design Methodology for Stealthy Parametric Trojans and Its Application to Bug Attacks,” in *CHES 2016*, ser. Lecture Notes in Computer Science, vol. 9813. Springer, 2016, pp. 625–647.
- [16] E. Biham, Y. Carmeli, and A. Shamir, “Bug Attacks,” in *CRYPTO 2008*,



**FIGURE 9:** 65 nm ASIC, PRNG on, clock frequency 25 MHz (Trojan not triggered),  $t$ -test results with 80 million traces (left), absolute maximum  $t$ -value over the number of traces (right).



**FIGURE 10:** 65 nm ASIC, PRNG on, clock frequency 50 MHz (Trojan triggered),  $t$ -test results with 80 million traces (left), absolute maximum  $t$ -value over the number of traces (right).



**FIGURE 11:** 65 nm ASIC, PRNG on, clock frequency 50 MHz (trojan triggered), CPA results targeting a key nibble based on an S-Box output bit with 40 million traces (right), absolute maximum correlation coefficient over the number of traces (left).

- ser. Lecture Notes in Computer Science, vol. 5157. Springer, 2008, pp. 221–240.
- [17] —, “Bug attacks,” *Journal of Cryptology*, vol. 29, no. 4, pp. 775–805, 2016.
- [18] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, “A Side-Channel Analysis Resistant Description of the AES S-Box,” in *FSE 2005*, ser. Lecture Notes in Computer Science, vol. 3557. Springer, 2005, pp. 413–423.
- [19] S. Mangard, N. Pramstaller, and E. Oswald, “Successfully Attacking Masked AES Hardware Implementations,” in *CHES 2005*, ser. Lecture Notes in Computer Science, vol. 3659. Springer, 2005, pp. 157–171.
- [20] D. Canright and L. Batina, “A Very Compact ‘Perfectly Masked’ S-Box for AES,” in *ACNS 2008*, ser. Lecture Notes in Computer Science, vol. 5037, 2008, pp. 446–459.
- [21] A. Moradi, O. Mischke, and T. Eisenbarth, “Correlation-Enhanced Power Analysis Collision Attack,” in *CHES 2010*, ser. Lecture Notes in Computer Science, vol. 6225. Springer, 2010, pp. 125–139.
- [22] S. Nikova, V. Rijmen, and M. Schläpfer, “Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches,” *J. Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.
- [23] C. Carlet, J. Danger, S. Guilley, and H. Maghrebi, “Leakage Squeezing of Order Two,” in *INDOCRYPT 2012*, ser. Lecture Notes in Computer Science, vol. 7668. Springer, 2012, pp. 120–139.
- [24] H. Maghrebi, S. Guilley, and J. Danger, “Leakage Squeezing Countermeasure against High-Order Attacks,” in *WISTP 2011*, ser. Lecture Notes in Computer Science, vol. 6633. Springer, 2011, pp. 208–223.
- [25] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup, “Threshold Implementations of Small S-boxes,” *Cryptography and Communications*, vol. 7, no. 1, pp. 3–33, 2015.
- [26] A. Poschmann, A. Moradi, K. Khoo, C. Lim, H. Wang, and S. Ling, “Side-Channel Resistant Crypto for Less than 2, 300 GE,” *J. Cryptology*, vol. 24, no. 2, pp. 322–345, 2011.
- [27] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, “Higher-Order Threshold Implementations,” in *ASIACRYPT 2014*, ser. Lecture Notes in Computer Science, vol. 8874. Springer, 2014, pp. 326–343.
- [28] T. Beyne and B. Bilgin, “Uniform First-Order Threshold Implementations,” in *SAC 2016*, ser. Lecture Notes in Computer Science, vol. 10532. Springer, 2017, pp. 79–98.
- [29] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, “A More Efficient AES Threshold Implementation,” in *AFRICACRYPT 2014*, ser. Lecture Notes in Computer Science, vol. 8469. Springer, 2014, pp. 267–284.
- [30] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, “Pushing the Limits: A Very Compact and a Threshold Implementation of AES,” in *EUROCRYPT 2011*, vol. 6632. Springer, 2011, pp. 69–88.
- [31] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, “Consolidating Masking Schemes,” in *CRYPTO 2015*, ser. Lecture Notes in Computer Science, vol. 9215. Springer, 2015, pp. 764–783.
- [32] H. Gross, S. Mangard, and T. Korak, “An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order,” in *CT-RSA 2017*, ser. Lecture Notes in Computer Science, vol. 10159. Springer, 2017, pp. 95–112.
- [33] A. Biryukov, C. D. Cannière, A. Braeken, and B. Preneel, “A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms,” in *EUROCRYPT 2003*, ser. Lecture Notes in Computer Science, vol. 2656. Springer, 2003, pp. 33–50.
- [34] D. Bozilov, B. Bilgin, and H. A. Sahin, “A Note on 5-bit Quadratic Permutations’ Classification,” *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 398–404, 2017.
- [35] A. Moradi and T. Schneider, “Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori,” in *ASIACRYPT 2016*, ser. Lecture Notes in Computer Science, vol. 10031, 2016, pp. 517–547.
- [36] A. Moradi and A. Wild, “Assessment of Hiding the Higher-Order Leakages in Hardware - What Are the Achievements Versus Overheads?” in *CHES 2015*, ser. Lecture Notes in Computer Science, vol. 9293. Springer, 2015, pp. 453–474.
- [37] P. Sasdrich, A. Moradi, and T. Güneysu, “Affine Equivalence and Its Application to Tightening Threshold Implementations,” in *SAC 2015*, ser. Lecture Notes in Computer Science, vol. 9566. Springer, 2015, pp. 263–276.
- [38] B. Bilgin, A. Bogdanov, M. Knezevic, F. Mendel, and Q. Wang, “Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware,” in *CHES 2013*, ser. Lecture Notes in Computer Science, vol. 8086. Springer, 2013, pp. 142–158.
- [39] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, “Threshold Implementations of All  $3 \times 3$  and  $4 \times 4$  S-Boxes,” in *CHES 2012*, ser. Lecture Notes in Computer Science, vol. 7428. Springer, 2012, pp. 76–91.
- [40] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. V. Assche, “Efficient and First-Order DPA Resistant Implementations of Keccak,” in *CARDIS 2013*, ser. Lecture Notes in Computer Science, vol. 8419. Springer, 2014, pp. 187–199.
- [41] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, “Trade-Offs for Threshold Implementations Illustrated on AES,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1188–1200, 2015.
- [42] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, “Suit up! - Made-to-Measure Hardware Implementations of ASCON,” in *DSD 2015*. IEEE Computer Society, 2015, pp. 645–652.
- [43] P. Sasdrich, A. Moradi, and T. Güneysu, “Hiding Higher-Order Side-Channel Leakage - Randomizing Cryptographic Implementations in Reconfigurable Hardware,” in *CT-RSA 2017*, ser. Lecture Notes in Computer Science, vol. 10159. Springer, 2017, pp. 131–146.
- [44] G. L. Smith, “Model for Delay Faults Based upon Paths,” in *International Test Conference 1985*. IEEE Computer Society, 1985, pp. 342–351.
- [45] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester, “Gate-length biasing for runtime-leakage control,” *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 8, pp. 1475–1485, 2006.
- [46] S. Eggersgluß, R. Wille, and R. Drechsler, “Improved sat-based atpg: More constraints, better compaction,” in *Proceedings of the international conference on computer-aided design*. IEEE Press, 2013, pp. 85–90.
- [47] “Genetic Algorithm,” <http://www.mathworks.com/discovery/genetic-algorithm.html>, [Accessed: 2016-02-01].
- [48] S. Ghandali, B. Alizadeh, and Z. Navabi, “Low Power Scheduling in High-level Synthesis using Dual-Vth Library,” in *16th International Symposium on Quality Electronic Design (ISQED)*, 2015, pp. 507–511.
- [49] X. Tang, H. Zhou, and P. Banerjee, “Leakage Power Optimization With Dual-Vth Library In High-Level Synthesis,” in *42nd annual Design Automation Conference (DAC 2005)*, 2005, pp. 202–207.
- [50] M. Ender, S. Ghandali, A. Moradi, and C. Paar, “The First Thorough Side-Channel Hardware Trojan,” in *ASIACRYPT 2017*, ser. Lecture Notes in Computer Science, vol. 10624. Springer, 2017, pp. 755–780.
- [51] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, “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](http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf).
- [52] T. Schneider and A. Moradi, “Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations,” in *CHES 2015*, ser. Lecture Notes in Computer Science, vol. 9293. Springer, 2015, pp. 495–513.
- [53] E. Prouff, M. Rivain, and R. Bevan, “Statistical Analysis of Second Order Differential Power Analysis,” *IEEE Trans. Computers*, vol. 58, no. 6, pp. 799–811, 2009.

- [54] S. Endo, Y. Li, N. Homma, K. Sakiyama, K. Ohta, D. Fujimoto, M. Nagata, T. Katashita, J. Danger, and T. Aoki, "A Silicon-Level Countermeasure Against Fault Sensitivity Analysis and Its Evaluation," *IEEE Trans. VLSI Syst.*, vol. 23, no. 8, pp. 1429–1438, 2015.



**Samaneh Ghandali** received the MSc degree in computer engineering from Shahid Beheshti University, Tehran, Iran, in 2009. Afterwards, till 2015 she worked as a graduate research assistant at the University of Tehran, Tehran, Iran. She is currently working toward the PhD degree in computer engineering under the supervision of Prof. Christof Paar at the University of Massachusetts, Amherst, USA. Her current research interest is hardware security with a special focus on physical security of embedded systems, hardware Trojans, side-channel analysis attacks and the corresponding countermeasures.



**M.Sc. Thorben Moos** received the B.Sc. and M.Sc. degrees in IT-Security from Ruhr-Universität Bochum, Germany in 2014 and 2016, respectively. Currently, he is a Ph.D. student and scientific research assistant at the chair for Embedded Security, Horst-Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany. His research interests include physical security of embedded devices with specialization in nano-scale side-channel analysis and secure ASIC implementation.



**Priv.-Doz. Dr. Amir Moradi** received the M.Sc. and Ph.D. degrees in computer engineering from Sharif University of Technology, Tehran, Iran, in 2004 and 2008 respectively. Afterwards, till 2015 he worked as a Post-Doctoral researcher at the chair for Embedded Security, Ruhr Universität Bochum, Germany. Since 2016, after obtaining the Habilitation degree, he has become a senior researcher and faculty member at the faculty of electrical engineering and information technology at Ruhr University Bochum. His current research interests include physical security of embedded systems, passive side-channel analysis attacks, and the corresponding countermeasures. He has published over 85 peer-reviewed journal articles and conference papers, in both destructive and constructive aspects of side-channel analysis. He also served as Program Committee Member (and the Chair) of several security- and cryptography-related conferences and workshops.



**Prof. Dr.-Ing. Christof Paar (Fellow, IEEE)** received his M.Sc. degree from the University of Siegen and the Ph.D. degree from the Institute for Experimental Mathematics at the University of Essen, both in Germany. He holds the Chair for Embedded Security at Ruhr University Bochum, Germany, and is Affiliated Professor at the University of Massachusetts Amherst, USA. He co-founded, with C. Koc, the Conference on Cryptographic Hardware and Embedded Systems (CHES). He has over 200 peer-reviewed publications and is coauthor of the textbook *Understanding Cryptography* (New York, NY, USA: Springer-Verlag, 2010). He is a co-founder of ESCRYPT – Embedded Security, a leading consultancy firm in applied security that is now part of Bosch. His research interests include the efficient realizations of cryptography, hardware Trojans, physical security and security evaluation of real-world systems.