TROJANUS : An ultra-lightweight side-channel leakage generator for FPGAs

Kutzner, Sebastian; Poschmann, Axel; Stöttinger, Marc

2013

Kutzner, S., Poschmann, A., & Stöttinger, M. (2013). TROJANUS: An ultra-lightweight side-channel leakage generator for FPGAs. 2013 International Conference on Field-Programmable Technology (FPT), 160 - 167.

<https://hdl.handle.net/10356/99561>

<https://doi.org/10.1109/FPT.2013.6718347>

© 2013 IEEE. This is the author created version of a work that has been peer reviewed and accepted for publication by 2013 International Conference on Field-Programmable Technology (FPT), IEEE. It incorporates referee's comments but changes resulting from the publishing process, such as copyediting, structural formatting, may not be reflected in this document. The published version is available at:
[DOI:<http://dx.doi.org/10.1109/FPT.2013.6718347>].

Downloaded on 26 Apr 2023 17:59:59 SGT

TROJANUS: An Ultra-Lightweight Side-Channel Leakage Generator for FPGAs

Sebastian Kutzner^{*†} and Axel Poschmann^{*†} and Marc Stöttinger^{*†},

^{*} Physical analysis and Cryptographic Engineering, Temasek Laboratories @ Nanyang Technological University

[†] Division of Mathematical Sciences, Nanyang Technological University, Singapore

{skutzner|aposchmann|mstottinger}@ntu.edu.sg

Abstract—In this article we present a new side-channel building block for FPGAs, which, akin to the old Roman god of Janus, has two contradictory faces:

- as a watermarking tool, it allows to uniquely identify IP cores by adding a single slice to the design;
- as a Trojan Side-Channel (TSC) it can potentially leak an entire encryption key within only one trace and without the knowledge of either the plaintext or the ciphertext.

We practically verify TROJANUS' feasibility by embedding it as a TSC into a lightweight FPGA implementation of PRESENT. Besides, we investigate the leakage behavior of FPGAs in more detail and present a new pre-processing technique, which can potentially increase the correlation coefficient of DPA attacks.

I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) combine the flexibility of micro controllers with the improved performance of Integrated Circuits (ICs), which make them often the preferred choice for embedded applications. To prevent Intellectual Property (IP) theft, the major FPGA vendors provide protection mechanisms, such as hard-macros and bit stream encryption [1]–[4]. Those implemented mechanisms shall prevent cloning and also to reconstruct embedded secrets in the design. However, Moradi et al. [5], [6] recently have broken the bit stream encryption with the help of side-channel analysis, this way enabling IP theft of full FPGA designs and product cloning. A potential technique to detect IP cores that have been disseminated unauthorizedly is watermarking, and side-channels based watermarking approaches have been proposed in [7]–[11]. The watermarking signatures get embedded in the power profile of a device in such a manner that the secret pattern is not obviously detectable, similar to the idea of image watermarking. The watermark pattern is usually generated by a special tailored circuit with a specific switching activity, which can be described by a so-called leakage function $\mathcal{L}_{\mathcal{F}}(\cdot)$. With the knowledge of $\mathcal{L}_{\mathcal{F}}(\cdot)$ and statistical analysis methods an evaluator can search for the watermarking pattern in the power profile and determine if the device is cloned or not.

The challenges to embedded a watermarking generating circuit are a) a low additional resource consumption and b) a good detectability when knowing the leakage function $\mathcal{L}_{\mathcal{F}}(\cdot)$, while hiding the signature of the watermark from unauthorized users, evaluators or attackers. Due to the current literature it seems that the usage of FPGA primitives to construct LFSRs with special sequence and combination functions seem

to be the most promising to construct such leaking watermark-circuits, cf. [10], [11]. In [7]–[9] the authors extend the usual design flow in Xilinx Inc. FPGAs to embed the LFSR construction using FPGA primitives. Our proposed TROJANUS mechanism can also be efficiently integrated into the design by using FPGA primitives, but it uses a more sophisticated scheme which results in a much smaller footprint while the detection gets more reliable. Identifying an IP core using, for instance a unique 20-bit ID (for the module or the customer) in combination with a 128-bit secret, can be implemented at the mere overhead of a single slice.

At the same time, this technique can also be abused to leak out secret information of a circuit. Customers who purchase an IP core, usually receive a hard-macro, that is a black-box, and hence, do not know what is *exactly* included in the IP core. It is thus hard to verify that an IP core does not contain malicious logic, called a *Hardware Trojan* (HT), added by an adversary. HTs can be classified using for example the taxonomy of [12], which defines three characteristics: physical, activation and action. The physical characteristic describes the type of manifestation of the HT, i.e., how it is realized, meaning, does it add gates, does it modifies existing wiring and so on. The activation characteristic describes how a Trojan is activated (externally or internally triggered) or if it is *always active*. The action characteristic specifies what the Trojan does once it is in its active state, e.g., leak information, modify the behavior of the system or completely destroy it, cf. [13], [14].

At CHES 2009 Lin et al. [15] presented the concept of *Trojan Side-Channels* (TSC), an *always active* HT which induces a physical side-channel (the power consumption) to leak information. They presented two designs: the first occupies 23 slices and can leak 80 out of 128 bits of an AES key, without known plaintexts. The second TSC presented occupies only 7 slices but needs known plaintexts. Both TSCs need 1030 measurements to recover the secret. In 2010 Gallais et al. [16] showed how HTs can induce or amplify side-channel leakage by a number of simple micro-architectural modifications. In 2012 Kasper et al. published a comprehensive article [11] on how to use side-channels deliberately in the design process, e.g., for watermarking [10], [17], but as well as for designing hardware Trojans. Their TSC requires 88 slices and leaks the masks of every measurement instead of the secret key, therefore reducing the probability of being detected by an

evaluator. However, they make some restrictive assumptions on the mask generation, i.e., they are generated by a rotating shift register. They need 500,000 clock cycles (in one or multiple measurements) to recover those masks which can subsequently be used in a standard CPA (Correlation Power Analysis).

TROJANUS, in its use for a malicious intent, falls in the category of TSC, hence it cannot be detected by functional testing which only checks for correctness using existing channels, e.g. I/O or special testing points. In addition we will show that even by standard side-channel measurements, as for example performed by evaluation labs for certification, it cannot be detected. Furthermore, since TROJANUS is very small, it is also very resistant to standard Trojan detection methods, because it does not change the propagation delay of the original circuit and has a very low signal-to-noise ratio in the overall circuit, cf. [14], [18], [19]. Last, TROJANUS can be used as a TSC for block ciphers and stream ciphers alike.

In the remainder of this article, in the experimental evaluation, we mainly focus on the HT aspects of TROJANUS. We chose to present Trojanus strength of leaking out information secretly, for instance watermarking IDs or secret keys, applied on a HT, because it is more challenging to extract unknown information from a side-channel leakage rather than binary falsify an known ID. We first present the basic idea of TROJANUS and provide some theoretical background. Afterwards, we practically verify TROJANUS as a TSC on an FPGA, discuss some problems and provide new solutions. We implemented two versions of the PRESENT block cipher [20]: an unsecured version and a second version secured by the Threshold Implementation (TI) countermeasure [21], [22]. Then we added TROJANUS to both designs. We show that it is possible to leak the whole key within one power trace (only capturing one encryption cycle) for the unsecured version and with only a few measurements for the secured version without either knowing the plaintext or ciphertext. Note that leaking the key within one trace is particularly interesting when attacking protocols where the key is updated in every encryption. Finally, we discuss open points and possible problems in practice and provide pointers for future work.

II. TROJANUS

A. Differential Power Analysis

Differential Power Analysis [23] exploits the fact that so-called side-channels, e.g. power consumption or electromagnetic emanation, depend on processed intermediate values. By statistically analyzing a lot of side-channel measurements, DPA is able to gain additional information about intermediate states and thereby reconstruct for example an encryption key of a cipher. CPA (Correlation Power Analysis) is a more advanced DPA technique introduced in [24]. The requirement [25] to perform a DPA/CPA is the need for an intermediate result which is a function of a known and changing value and part of the unknown fixed secret. A classic example for a suitable intermediate result is the Sbox output of the first round of a block cipher. Since our leakage generator cannot

fulfill this requirement, we introduce a new methodology to leak static values without known input, but which can still be retrieved by a DPA-like attack.

B. Concept

As mentioned in the last section our leakage generator does not meet the classic requirements for a DPA, i.e., we cannot target an intermediate result which is a function of a known changing input and the secret. But the requirement for a DPA can be more generalized as follows: one must be able to target a predictable changing output, which *somehow* depends on the secret. So the new question is: *how to leak a static key without additional known input?* Our idea is to use a changing but predictable (for the attacker/evaluator) leakage function $\mathcal{L}_{\mathcal{F}_t}(\cdot)$ with a static input, e.g., the secret key or the watermarking ID.

Compared to the approach published in [11] we do not need to know a second parameter to generate a predictable leakage. The leakage function of Kasper et al. can be described for ASIC and FPGA implementations as follows:

$$\mathcal{L}_{\mathcal{F}_t, HT_{ka}}(k_i, s) = F(k_i, s) \oplus F(k_i, s - 1), \quad (1)$$

where k_i denotes the byte of the round key under attack and s the current state of an LFSR, while $(s - 1)$ denotes its previous state. Ziehner et al. in [8] uses LFSR primitive FPGA components in a similar manner to embed a watermarking signature in the power consumption profile. Note that both s and $(s - 1)$ are only known to the attacker and thus constitute a secret only known to the attacker. Both parameters are used by a fixed combination function $F(\cdot, \cdot)$ to generate a hidden side-channel leakage. To hamper detection of the key dependent leakage it is possible to extend the combination function to use the plaintext, cf. [15]. Another proposed solution to hamper detection is to leak the masks of a secured implementation instead, which enables the attacker to subsequently perform a DPA with known masks, cf. [11].

Our proposed scheme is independent of the plaintext or the masking value. It extends the idea proposed in [9] by not only adding a fixed LFSR scheme in the unused space of a LUT, instead we are modulating the leakage generating function itself and, thus, the scheme only depends on the static value that shall be leaked (e.g. key or ID) as the sole parameter of $F(\cdot)$:

$$F_{(m)}(k_i) = \{f_1(k_i), f_2(k_i), f_3(k_i), \dots, f_m(k_i)\}. \quad (2)$$

The i -th function f_i is selected by a discrete time value, such as a certain clock cycle, in order to change the leaked output over time instead of using a second parameter, such as the plaintext. The leakage is the distance of two consecutive function outputs and can thus be modeled as:

$$\mathcal{L}_{\mathcal{F}_t, HT}(k_i; t) = f_t(k_i) \oplus f_{t-1}(k_i). \quad (3)$$

The observable side-channel leakage $\mathcal{L}_t$ caused by the leakage generator is therefore a composition of the leakage caused by the original design $\mathcal{L}_{\mathcal{F}_t, cipher}(k_i, x_i)$ and the embedded leakage generator $\mathcal{L}_{\mathcal{F}_t, HT}(k_i; t)$:

Fig. 1. Chain of CFGLUT5s

$$\mathcal{L}_t \leftarrow HW(\mathcal{F}_{t,cipher}(k_i, x_i)) \circ HW(\mathcal{F}_{t,HT}(k_i; t)). \quad (4)$$

In case the Trojan is synchronized with the start of each operation and the unique sequence of the leakage generator is smaller or equally long as the operation, the generated leakage will be the same at any fixed point in time over multiple operations. Hence, Eq. (4) can be simplified too:

$$\mathcal{L}_t \leftarrow HW(\mathcal{F}_{t,cipher}(k_i, x_i)) \circ HW(const.) \quad (5)$$

for each observation at point t_0 and the leakage cannot be exploited by a standard DPA.

C. The Side-Channel Leakage Generator

Fortunately, Xilinx Virtex-5 FPGAs offer the perfect building block for our intend, a special Look-Up Table (LUT) called *CFGLUT5*.¹ This element is an at runtime dynamically reconfigurable 5-input to 1-output Look-Up Table that enables changing the logical function of the LUT during circuit operation [28]. The 32-bit sequence (2^5) which defines the logical function of the LUT can be shifted by one bit in every clock cycle (internally implemented as a shift register), thus changing the output function and, hence, also the leakage, in every clock cycle. As a consequence, every clock cycle can now be seen as one leakage trace for a DPA. Furthermore, the bit which falls out of the configuration/shift register can be used as input to another *CFGLUT5*, thereby allowing a serial chain of data to reconfigure multiple LUTs, see Fig. 1. We will explain our implementation in more detail in the next section.

Since the *CFGLUT5* has a 5-bit input, we can leak 5 bits per *CFGLUT5*. To leak an n -bit secret we therefore need $\lceil \frac{n}{5} \rceil$ *CFGLUT5*s. That means we get $\lceil \frac{n}{5} \rceil * 32$ (potentially) different leakage functions or in other words, a leakage sequence of $\lceil \frac{n}{5} \rceil * 32$ bits length. How to choose the initial state of our *CFGLUT5*s will be discussed later in Section IV-A.

D. Practical Implementation

As described in the introduction, TROJANUS can be used efficiently for watermarking. Assume that a vendor wants all of his IP cores to have a unique identifier, he first chooses a 20-bit value, meaning the side-channel generator would consist of $20/5 = 4$ *CFGLUT5*s, which fit in only one slice. The sequence defining the LUT functions has a length of $4 * 32 = 128$ bits, hence, the complete ID for this core would have a length of 128 bits. The vendor chooses a fixed 128-bit sequence (or a set of sequences), representing the vendor's

master key. Using this master key, a vendor can test a product for IP infringement by recovering the 20-bit ID. In a similar procedure TROJANUS can be "abused" as a SCT leaking out secrets. Instead of using a fixed and known ID a suspicious designer can chose a fixed secret sequence to recover for instance the content of the key register connoted to the input of the *CFGLUT5*s of the TROJANUS scheme.

Hence, we provide an example of applying TROJANUS as a SCT embedded in a PRESENT block cipher and show how efficient a unknown secret can be leakage out secretly. First we describe our implementation, then we show how to exploit the leakage caused by our leakage generator, before we discuss how to reconstruct the secret key. As mentioned before we focus on TROJANUS as a HT here, but all results can be easily transferred for its use as a watermarking tool, i.e., leaking an ID instead of a secret key.

For our practical evaluations we chose the PRESENT cipher [20], one unprotected version (**P1**) and one protected with the TI [21] countermeasure (**P2**), both using an 80-bit key, cf. [22]. Figure 2 shows the implementation of our HT based on TROJANUS embedded in an unprotected serialized PRESENT implementation. As described in the previous section, we need $\lceil \frac{n}{5} \rceil$ *CFGLUT5*s, meaning $\lceil 80/5 \rceil = 16$ LUTs, to leak the whole key. The inputs for the 16 *CFGLUT5*s are directly taken from the key register of the interface (since the key register of the cipher itself is updated every round). No functional changes are made to the original cipher and no additional logic is inserted in the cipher, which could cause timing or delay differences. This is why the proposed TSC is easily to integrate in any existing circuit.

Our PRESENT implementation takes 516 clock cycles for one encryption. Since every clock cycle represents one leakage trace, we theoretically can obtain 516 traces for a DPA from one encryption. Our sequence is $16 * 32$ bits long, resulting in 512 different leakage functions, hence perfectly fitting in one encryption trace. In a noisy environment where more traces are needed to perform a DPA, it is easy to measure several encryptions and average the measured traces before performing a DPA. On the other hand, if the cipher has less clock cycles than the sequence, the sequence is split over consecutive encryptions and multiple measurements have to be cut together. It is even possible to keep the TSC running

Fig. 2. TROJANUS embedded as a TSC in a lightweight PRESENT implementation **P1**.

¹Note that other vendors offer similar building blocks, cf. [26], [27], in their FPGAs, hence, our design can be transferred to other FPGA platforms with only small modifications.

Fig. 3. CPA Results

independently of the encryption core. Note that in our design the Trojan circuit is only active for 512 clock cycles to ease synchronization in the attack phase, but this is not necessary for a successful attack. A more thorough discussion about synchronization and averaging will be given in Section IV.

As for the size, one slice on a Virtex-5 can host 4 *CFGLUT5s* and 4 flip-flops to store the output bits of the four LUTs, resulting in a total size of only 4 slices for a TROJANUS that can leak out an entire 80-bit PRESENT key. A visualization of the resource consumption for leaking various key lengths is provided in the Appendix, cf. Fig. 8.

III. SIDE-CHANNEL ANALYSIS

A. Measurement Setup

Our power consumption traces were obtained from a SASEBO G-II evaluation platform. The SASEBO G-II hosts two FPGAs, i.e., a control FPGA (Xilinx XC3S400A-4FTG256, Spartan-3A series) and a cryptographic FPGA (Xilinx XC5VLX50-1FFG324, Virtex-5 series) which is decoupled from the rest of the board in order to minimize electronic noise from the surrounding components. It is supplied with a voltage of 1V by an external stabilized power supply as well as with a 3 MHz clock derived from the 24 MHz on-board clock oscillator. The power consumption is measured over a $1\ \Omega$ resistor inserted in the VDD line by using a differential probe. All power traces are collected with a LeCroy WR610Zi-s-32 oscilloscope at a sampling rate of 10 GS/s.

B. FPGA Characteristics

Our first experiments were performed with a design that includes all 16 *CFGLUT5s*, but only one output bit is connected to an output driver to mimic a load, i.e., the output bit of the function for which the most significant five key bits are the input. With this test we evaluate our idea to leak five static key bits with a changing but predictable 5-to-1 bit function. One trace consisting of 512 clock cycles² was recorded and then sliced into 512 single traces, each trace containing one clock cycle.

²Maximum cycle count of our bit stream defining the outputs of the 16 *CFGLUT5s*, the last 2 two clock cycles of the encryption were ignored

Since we know the starting value as well as the sequence of the 16 *CFGLUT5s* 5-to-1 bit functions, we can calculate the output of the first function for all 512 clock cycles. To calculate the 32 (2^5) hypothetical power values for our CPA we chose the Hamming distance between two consecutive outputs of our target function, cf. Eq (3). Figure 3(a) depicts the result of the CPA. As we can see the correct key hypothesis does not stand out and is therefore not clearly identifiable, in contrary to our expectations.

Consequently, we had to analyze the leakage behavior in more detail to reveal the reason for the failed CPA. Figure 4(a) shows three power traces, one representing a 1-to-0 transition of the output bit (Hamming distance = 1), one representing a 0-to-1 transition (Hamming distance = 1) and the last one representing no transition, i.e. a 0-to-0 transition (equal to a 1-to-1 transition, Hamming distance = 0). As we can see in Fig. 4(a) there is a clear difference between a transition and no transition which should enable a successful CPA. But we also see that the two transitions show a complementary behavior which causes them to cancel each other out in a CPA

Fig. 4. FPGA leaking power consumption

(a) First 5-bit chunk of key

(b) 11th 5-bit chunk of key

Fig. 5. Results of CPA only on the isolated Trojan circuit

(a) First 5 bit

(b) Second 5 bit

Fig. 6. Results of CPA on Trojan circuit injected in PRESENT cipher **P1**

(a) Standard CPA on protected PRESENT

(b) Results of CPA exploiting the Trojanus leakage

Fig. 7. Results of CPA on protected PRESENT cipher **P2** with injected Trojan circuit

and decrease the resulting correlation coefficient. This can be easily deduced from Eq. (6), despite the clear differences.

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (6)$$

A straightforward solution is to use three model values for the CPA, e.g., -1 , 0 and 1 , representing the three different transitions. Figure 3(b) shows the result of a CPA using the latter model. One can clearly see that the CPA works better as before and it is now possible to identify the correct key hypothesis, because the correct key has slightly the absolute maximum value.

To prevent that the power traces cancel each other out in a CPA we propose to pre-process them in the following way. We calculate the mean of each sample and use this resulting mean trace as *mirror* since the mean trace closely resembles a trace representing a *no transition* (cf. Fig. 4(b)). All signals below that mean trace are mirrored to the opposite site, such that all samples are above that mean trace. With the resulting traces a standard CPA is performed again with the classical Hamming distance model as already performed above. Figure 3(c) depicts the resulting correlation trace after the proposed pre-processing step. As we can see the correlation for the correct hypothesis clearly stands out and is roughly three times as high as a CPA with the 3-value model.

An open question is, what causes this special leakage behavior. We ran some tests analyzing the leakage of a single flip-flop, with and without a load. Without a load the flip-flop

showed only minimal differences in the power consumption between transition and no transition. When connecting an output driver to mimic a load on the other hand, we could see the same complementary leakage behavior as described above. Unfortunately we did not have the time to test different building blocks to mimic a load, but we assume that the type of load can not only influence the strength of the leakage, but also its form. This circumstance of course could also be used to further hamper the detection of the Trojan since the standard side-channel models do not fit and its leakage is disguised. We will investigate this manner in future work.

C. Leaking Secret Bits with one Measurement

We started by implementing only the TSC circuit, i.e., all 16 output bits are now connected to a load and are leaking the whole key in parallel, but no PRESENT is in place yet. The process is again the same. We recorded one power trace containing all 512 clock cycles of a full sequence. Then, the trace is sliced in 512 single traces and a CPA is performed (including pre-processing). Figure 5 shows two exemplary results of a CPA attacking the first 5-bit chunk of the key and the 11th 5-bit chunk, respectively. As one can see, the correct key hypotheses are clearly distinguishable.

Next we embedded TROJANUS as a TSC in an unprotected PRESENT core, see Fig. 2. The procedure is rather the same than adding a secret watermark to the PRESENT cipher. Instead of connecting the register output of the key register with the CFGLUT5 input the output of the register storing the watermarking value is connected with the CFGLUT5 inputs

Again, the same attack as described before is performed and Fig. 6 shows the results of the CPA. As we can see it is still possible to retrieve the correct key with only one trace. One could question the significance of this experiment because an unprotected PRESENT can be also easily attacked with a standard CPA. This is of course true, but there exist protocols where a new session key is derived for every encryption. There were also side-channel countermeasures proposed where a new key is used for every encryption [29]. These are scenarios where the possibility to extract the key from an unprotected circuit with one measurement is very interesting. Also the application on public key encryption schemes seems a suitable scenario to extract the secret within one encryption run.

Finally, we embedded TROJANUS as a TSC in a protected implementation of the PRESENT cipher. This time, it was not possible anymore to retrieve the key within a single trace due to the higher noise introduced by the PRESENT circuit working in parallel. But averaging 1,000 traces is already enough to retrieve the key again (less traces are probably sufficient as well), as can be seen in Fig. 7(b). Note that for this last measurement we changed the clock to 4 MHz so we can split the traces more easily, i.e. one clock cycle consists of 2500 samples now (instead of $3,333.\bar{3}$ samples).

To prove that the proposed TSC does not cause unwanted leakage which may reveal the Trojan during an evaluation process, we performed a standard CPA against the protected PRESENT implementation as described in [22]. In our implementation of the Trojan the sequence starts the same time an encryption starts. This means that at every position in time the Trojan circuit has the same state during multiple measurements, causing the same leakage in every trace. Hence, since a DPA analyzes every point in time independently and the TSC causes constant leakage at every point in time over multiple measurements, cf. Eq. (5), there is no leakage, which can be detected by standard evaluation methods, see Fig. 7(a). However, there are methods an evaluator could use if he was specifically looking for this kind of Trojan. This will be discussed in more detail in Section IV.

IV. DISCUSSION

In this section we would like to discuss some open questions regarding the design TROJANUS as well as different approaches to perform the side-channel analysis depending on available conditions.

A. Choice of Sequence

At first, we investigate how to choose the sequence defining the output functions for the *CFGLUT5s* to maximize the leakage. We tried different sequences, e.g., repeating 32-bit patterns, specially crafted patterns, which assures that each *CFGLUT5* has a unique pattern, like 010100110011000111... and so on, and investigated which of these patterns maximizes the leakage. In the end it turned out that a completely random sequence yielded the best results. As future work we want to investigate if there exist a better sequence in more detail.

TABLE I
COMPARISON OF TROJAN SIDE CHANNELS.

Applied on	Ref.	Leakage	Traces	Slices
Sole running	This paper	20 bits	1 ¹	1
Sole running	This paper	80 bits	1 ¹	4
Sole running	This paper	128 bits	1 ¹	7
PRESENT P1	This paper	80 bits	1 ²	4
AES	Lin et al. [15]	80 bits	1,030	23
AES	Lin et al. [15]	80 bits	1,030	7
TI PRESENT P2	This paper	80 bits	1,000	4
Boolean masked AES	Kasper et al. [11]	128 bits	1 ³	88

¹ Trace contains at least only 32 clock cycles.

² Trace contains exactly 512 clock cycles only.

³ Trace contains at least 500,000 clock cycles.

We also tried to directly feedback the output of every LUT back to its input. This would reduce the sequence length to 32 but would probably give an attacker more freedom in the place&route of the leakage generating TROJANUS circuit. We repeated the attack but we were not able to retrieve any of the key bits. We assume that the short sequence of 32 bit is not enough to differentiate all 32 key hypotheses from each other, which implies that, the longer the sequences, the stronger the attack.

B. Leaked Data Recovery

Second, we would like to discuss the different approaches for performing the leaked data recovery. The attacks are strongly dependent on three factors: activity, synchronization, and length of activity. Is the Trojan only active during encryption or is it always active? On the one hand, if it is only active during encryption it is easier for an attacker to synchronize the measurements, but on the other hand he might have to execute multiple encryptions to get enough information for a CPA. Especially if the encryption takes only a few cycles, an attacker will have to capture multiple measurements. In this case the attacker has to know exactly for how many cycles the Trojan is active to derive the state of the sequence for the next measurement. Otherwise he is not able to 1) shift every measurement such that sequences match and he is able to average the correct parts of the trace; 2) calculate the correct hypothesis for the CPA when analyzing multiple measurements.

If the Trojan is always active, an attacker just has to measure the power consumption of the chip at an arbitrary point in time, but preferably one where the rest of the chip is idle to reduce noise and ease the attack.

No matter if the Trojan is always active or only active during encryption for a certain (known) amount of clock cycles, in both cases the attacker has to guess the initial state of the sequence. But since he knows the sequence he just has to perform *length of sequences* (2^5) times a CPA for every possible rotation of the state. With the knowledge of the initial state he is then able to perform all subsequent attacks.

C. Performance and Comparison

Please note that due to the lack of an existing figure of merit, a comparison between different HT designs is not an easy

TABLE II

ESTIMATED AREA OVERHEAD OF OUR HT TO LEAK THE FULL KEY OF SELECTED BLOCK AND STREAM CIPHERS SYNTHESIZED ON A VIRTEX-5 PLATFORM. THE RELATIVE, ADDITIONAL RESOURCE OVERHEAD FOR EMBEDDING TROJANUS (Δ HT) IS BASED ON THE SLICE CONSUMPTION.

Algorithm	Key length	HT [Slices]	Reference	Path [bits]	Resources			Δ HT [%]
					[LUT]	[Slices]	[FF]	
TRIVIUM	80	4	unprotected [30]	1	359	237	242	1.7
PRESENT	80	4	Boolean masking [31]	64	753	263	282	1.5
			TI lightweight [22]	4/64	360	112	311	3.4
AES	128	7	MDPL [32]	128	12,104	4,744	1,196	0.1
			TI [33]	128	12,597	4,642	1,295	0.2
			WDDL [34]	128	7,265	2,590	1,160	0.3
			MAO [35]	128	3,880	1,633	673	0.4
			Masked Canright [36]	128	3,626	1,459	557	0.5
			LUT masking [37]	128*	4,772*	1,462*	904*	0.5

* Numbers are taken directly from publication.

task. However, all designs leak some bits, require a certain number of traces or clock cycles, and occupy a specific number of slices, which we will use for comparison. TROJANUS has a better trade-off in terms of resource consumption and leaking bits per encryption compared to other published TSC designs [11], [15], [16], cf. Table I. Due to TROJANUS' compact structure and efficient bit leaking scheme, it is very hard to detect and therefore even suitable for lightweight implementations. Furthermore, in an optimal case it is possible to leak the complete key in only one trace (see Sect. III-C), while the other published TSC schemes need more traces (or one very long trace covering 500,000 clock cycles [11]) to leak the complete key.

Table II provides an estimation of the additional resources needed to embed TROJANUS as a TSC in various SCA-protected block cipher and stream cipher implementations. We have taken the implementations of the corresponding references and synthesized them on a Virtex-5 using Xilinx ISE 13.3. The overhead for embedding TROJANUS, Δ HT, is based on the slice consumption. For example, a HT leaking the entire 128-bit AES key out of an MDPL-protected implementation requires 7 slices, resulting in an overhead of only 0.1%.

TROJANUS was designed for FPGAs because they offer fancy building blocks which can perfectly map our idea to an implementation. However, it is also possible to map the concept of TROJANUS to an ASIC design with some additional costs by using linear shift register and multiplexer components instead of the CFGLUT5 building block.

D. Detectability

In [11], [15], [16] the authors describe an approach on how to detect Side-channel Trojans. The idea is to take multiple measurements with fixed plaintexts but two different keys and compare the average, e.g., power consumption, of these two sets. A difference should be visible, because the Trojan strongly depends on the key. Similar assumption can also be made for the imprinted watermarking signature in the power consumption using TROJANUS.

Again, we would like to discuss two different scenarios: Synchronized and non-synchronized. In the synchronized case

the leakage generator TROJANUS is only active during encryption. But since the encryption algorithm obviously depends on the key as well, the power characteristics of the algorithm itself will also change. Hence, differences in the power consumption cannot be affiliated to TROJANUS. In the unsynchronized case on the other hand, an evaluator is not able to build the needed averages in the first place because the power consumption signature of TROJANUS is different in every measurement and any information about potential differences is destroyed. Summarizing, our leaking generator TROJANUS is immune against this kind of detection method. Hence, the imprinted watermarking ID adds no suspicious characteristics in the power consumption, which can be easily detected or extracted. Furthermore, Using Trojanus as a SCT, most standard detection methods, e.g. as described in [14], [18], [19], will most likely fail as well, because of its small size.

V. CONCLUSION

In this paper we introduced TROJANUS, a new ultra-lightweight side-channel leakage generator specially tailored for FPGAs. It can be used to construct both, Trojan side-channels as well as watermarking schemes for arbitrary FPGA designs with a negligible overhead. The core element of this new versatile building block is a dynamically reconfigurable 5-to-1 LUT, which is used to change the leakage function for each clock cycle. We introduced a scheme in which the information to be leaked is transmitted over multiple clock cycles in a unique, non-public sequence of leakage functions, which ensures a hidden and hard to detect side-channel leakage and thus provide a perfect utility to implement a TSC or hide watermarking signatures in the power profile.

In order to verify the feasibility of TROJANUS in practice, we have conducted several experiments and were able to reconstruct unknown leaked secrets from power measurements using TROJANUS as a Hardware Trojan. We were able to leak the entire 80-bit secret key of an unprotected PRESENT implementation with only one measured encryption. This can be particularly interesting to overcome protocols or countermeasures that update an encryption key for every execution. Second, we have shown that the key of a protected PRESENT implementation can be leaked with only 1000 traces by

embedding TROJANUS as a TSC, while standard side-channel analysis methods fail to detect it.

REFERENCES

- [1] R. Krueger, "Application Note XAPP766: Using High Security Features in Virtex-II Series FPGAs," in *Technical report, Xilinx*, available via http://www.xilinx.com/support/documentation/application_notes/xapp766.pdf, 2004.
- [2] C. W. Tseng, "Lock Your Designs with the Virtex-4 Security Solution," in *XCell Journal, Xilinx*, 2005.
- [3] Altera, "AN 341: Using the Design Security Feature in Stratix II and Stratix II GX Devices," in *Technical report, Altera*, available via <http://www.altera.com/literature/an/an341.pdf>, 2009.
- [4] MicroSemi, "The Importance of Design Security," in *Technical report, Microsemi*, available via http://www.microsemi.com/document-portal/doc_download/125154-the-importance-of-design-security-brochure.
- [5] A. Moradi, A. Barengi, T. Kasper, and C. Paar, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys From Xilinx Virtex-II FPGAs," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 111–124.
- [6] A. Moradi, M. Kasper, and C. Paar, "Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures," in *Topics in Cryptology—CT-RSA 2012*. Springer-Verlag, 2012, pp. 1–18.
- [7] D. Ziener and J. Teich, "FPGA Core Watermarking Based on Power Signature Analysis," in *FPT*, G. A. Constantinides, W.-K. Mak, P. Sirisuk, and T. Wangtong, Eds. IEEE, 2006, pp. 205–212.
- [8] —, "Power Signature Watermarking of IP Cores for FPGAs," *Signal Processing Systems*, vol. 51, no. 1, pp. 123–136, 2008.
- [9] M. Schmid, D. Ziener, and J. Teich, "Netlist-level IP Protection by Watermarking for LUT-Based FPGAs," in *FPT*, T. A. El-Ghazawi, Y.-W. Chang, J.-D. Huang, and P. Saha, Eds. IEEE, 2008, pp. 209–216.
- [10] D. Ziener, F. Bauerregger, and J. Teich, "Multiplexing Methods for Power Watermarking," in *HOST*, J. Plusquellic and K. Mai, Eds. IEEE, 2010, pp. 36–41.
- [11] 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.
- [12] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.
- [13] Y. Jin, N. Kupp, and Y. Makris, "Experiences in Hardware Trojan Design and Implementation," in *HOST*, M. Tehranipoor and J. Plusquellic, Eds. IEEE, 2009, pp. 50–57.
- [14] Y. Jin and Y. Makris, "Hardware Trojans in Wireless Cryptographic ICs," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.
- [15] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, "Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering," in *CHES*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 382–395.
- [16] J.-F. Gallais, J. Großschädl, N. Hanley, M. Kasper, M. Medwed, F. Regazzoni, J.-M. Schmidt, S. Tillich, and M. Wójcik, "Hardware Trojans for Inducing or Amplifying Side-Channel Leakage of Cryptographic Software," in *International Conference on Trusted Computing*, L. Chen and M. Yung, Eds. Springer, 2010.
- [17] G. T. Becker, M. Kasper, A. Moradi, and C. Paar, "Side-channel based Watermarks for Integrated Circuits," in *HOST*, J. Plusquellic and K. Mai, Eds. IEEE, 2010, pp. 30–35.
- [18] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 296–310.
- [19] Y. Jin and Y. Makris, "Hardware Trojan Detection Using Path Delay Fingerprint," in *HOST*, M. Tehranipoor and J. Plusquellic, Eds. IEEE, 2008, pp. 51–57.
- [20] 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*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 450–466.
- [21] S. Nikova, V. Rijmen, and M. Schläffer, "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches," *Journal of Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.
- [22] A. Poschmann, A. Moradi, K. Khoo, C.-W. 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.
- [23] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proceedings of CRYPTO 1999*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.
- [24] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES*, ser. Lecture Notes in Computer Science, M. Joye and J.-J. Quisquater, Eds., vol. 3156. Springer, 2004, pp. 16–29.
- [25] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.
- [26] MicroSemi, "SmartFusion FPGA Fabric - Users Guide." [Online]. Available: Available via http://www.actel.com/documents/SmartFusion_Fabric_UG.pdf
- [27] Altera, "lpm_shiftreg Megafunction." [Online]. Available: Available via http://www.altera.com/literature/ug/lpm_shiftreg.pdf
- [28] Xilinx, "Virtex-5 Libraries Guide for HDL Design." [Online]. Available: Available via <http://pdfsb.com/virtex+5+libraries+guide>
- [29] M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni, "Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices," in *AFRICACRYPT*, ser. Lecture Notes in Computer Science, D. J. Bernstein and T. Lange, Eds., vol. 6055. Springer, 2010, pp. 279–296.
- [30] C. D. Cannière and B. Preneel, "Trivium," in *The eSTREAM Finalists*, ser. Lecture Notes in Computer Science, M. J. B. Robshaw and O. Billet, Eds. Springer, 2008, vol. 4986, pp. 244–266.
- [31] M. Stoettinger, T. Feller, and S. A. Huss, "A Side-Channel Hardened IP-Protection Scheme for FPGA-based Platforms," in *FPT*, R. Tessier, Ed. IEEE, 2011.
- [32] T. Popp and S. Mangard, "Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints," in *CHES*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 172–186.
- [33] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," in *ICICS*, ser. Lecture Notes in Computer Science, P. Ning, S. Qing, and N. Li, Eds., vol. 4307. Springer, 2006, pp. 529–545.
- [34] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *DATE*. IEEE, 2004, pp. 246–251.
- [35] E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data," *Cryptology ePrint Archive*, Report 2003/236, 2003.
- [36] D. Canright and L. Batina, "A Very Compact "Perfectly Masked" S-Box for AES," in *ACNS*, ser. Lecture Notes in Computer Science, S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, Eds., vol. 5037. Springer, 2008, pp. 446–459.
- [37] F. Regazzoni, W. Yi, and F.-X. Standaert, "FPGA Implementations of the AES Masked Against Power Analysis Attacks," in *COSADE*, 2011, pp. 55–66.

APPENDIX

Figure 8 depicts the resources needed for leaking different key lengths using the CFGLUT5 component of the Virtex 5 architecture.

Fig. 8. Resource consumption depending on key length.