Generic Side-Channel Countermeasures for Reconfigurable Devices*

Tim Güneysu and Amir Moradi

Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{gueneysu,moradi}@crypto.rub.de

Abstract. In this work, we propose and evaluate generic hardware countermeasures against DPA attacks for recent FPGA devices. The proposed set of FPGA-specific countermeasures can be combined to resist a large variety of first-order DPA attacks, even with 100 million recorded power traces. This set includes generic and resource-efficient countermeasures for on-chip noise generation, random-data processing delays and S-box scrambling using dual-ported block memories. In particular, it is possible to build many of these countermeasures into a single IP-core or hard macro that then provides basic protection for any cryptographic implementation just by its inclusion in the design process – what is particularly useful for engineers with no or little background on IT security and SCA attacks.

1 Introduction

Since the last fifteen years, side-channel analysis (SCA) [12] attacks have been (publicly) known as a major threat to any unprotected cryptographic implementation in software and hardware. Lots of efforts have already been dedicated towards the development of corresponding countermeasures, in particular against differential power analysis (DPA) [13], such as [5, 11, 14, 15, 18–21, 23, 24] with this list far from being complete. A particular subject of study has been on algorithmic countermeasures that mask or shuffle security-critical processes of a specific cryptographic system as well as on generic hardware countermeasures, such as noise generators, non-deterministic processors or side-channel resistant logic styles. Based on all these observations it has widely been accepted that a single (and efficient) countermeasure cannot provide complete protection against a large variety of SCA attacks. Hence, a mix of several countermeasures is typically required to provide the security as demanded by the protection profile for a given application (e.g., dictated by an untrusted operation environment and the attacker model). Despite the information theoretic metrics defined by [22], the resistance for such a protection profile, such as against DPA attacks, is typically specified by a number of samples that need to be recorded for a successful attack, as typically done in common criteria evaluations. In other words, if all (known)

* The work described in this paper has been supported by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

SCA attacks with a given number of recorded samples fail, the device is supposed to be sufficiently resistant according to the specified protection profile [16].

In this context, it is primarily the decision of the system designer which combination of countermeasures should be implemented in a device. Unfortunately, choosing a suitable combination of countermeasures is a rather challenging and tedious task in practice. In particular, each device that uses a different technology, architecture or combination of several countermeasures may behave differently. In this context, a hardware developer (in particular one with no or little background on cryptography and/or SCA) would be pleased to have a set of generic, cheap and pre-evaluated hardware-countermeasure implementations at hand that can be easily integrated and combined to achieve a product-specific protection profile.

In this work, we propose and evaluate generic hardware countermeasures against DPA attacks which are suitable for a large variety of recent FPGA devices and cryptographic systems (in particular, devices from Xilinx/Altera). Recently, FPGAs are commonly used for many cryptographic implementations and provide a multitude of different pre-fabricated hardware resources. We will evaluate the most promising reconfigurable resources concerning their usability for generic DPA countermeasures. We finally present resource efficient and generic design approaches for DPA countermeasures that can be combined to resist a large variety of first-order DPA attacks. This includes countermeasures for on-chip noise generation, the insertion of random data processing delays and memory scrambling. In particular, it is possible to combine many of these evaluated countermeasures into a single hard macro to achieve basic protection for any cryptographic implementation – just by its inclusion into the synthesis process. Such an available IP core can be especially valuable for engineers with no or little background on side-channel analysis and/or cryptography.

This work is structured as follows. Section 2 introduces different novel and generic countermeasures which are built from the available resources of recent FPGA devices. In Section 3 we briefly introduce an unprotected AES design as reference implementation for our evaluation. Our measurement setup and the evaluation method used to analyze the impact of each proposed countermeasure are also provided as part of this section. Our results are presented in Section 4 before we conclude in Section 5.

2 Generic Countermeasures for FPGAs

It is well known that generic hardware countermeasures against DPA attacks primarily need to decrease the relation between data processed by a relevant part of the cryptographic implementation and the actual power consumption of the device. There are several options to achieve this goal:

- *Reducing the Signal-to-Noise Ratio*: An attacker attempts to exploit a specific part D_t of a power trace P_t that processes key-dependent data within a (known) cryptographic implementation at a given point in time t. A straightforward countermeasure is thus to bury D_t with lots of additive (Gaussian)

noise N_t so that the overall power trace can be modeled as $P_t = D_t + N_t$. It is evident that the addition of noise is not capable to hide the attackable part D_t completely but it can complicate a practical DPA attack, especially when combined with further countermeasures.
- *Timing Disarrangement*: DPA attacks operate on a high number of (key-dependent) data points that are assumed to be sampled at exactly the same point in time. The attacker usually runs a series of alignment filters to overcome any intrinsic misalignment within the data processing, e.g., due to clock jitter or other operational variations. An effective countermeasure is to further randomize or shuffle the points in time when such attackable operations are processed. Of course, this method can also be overcome [3, 5], requiring the attacker to use advanced filtering functions beyond simple peak alignment, such as complex integration and windowing methods.
- *Signal Masking and Scrambling*: An extensive protection against DPA can only be obtained when the attackable part of the signal D_t completely disappears in the power trace. This can be done by applying random masks to D_t. Unfortunately, this strategy is usually very specific, requires expert knowledge and involves significant changes of the cryptographic algorithm at the additional cost of reduced performance and increased resource consumption (see, e.g., for rather costly proposals [7, 8]).

In the next subsections we investigate implementation strategies for generic FPGA countermeasures that are widely available for a large number of (symmetric) cryptographic systems. In particular, a primary design goal for this analysis will be to utilize only the (limited amount of) prefabricated resources that are available on recent (Xilinx/Altera) FPGAs.

2.1 Generating Gaussian Noise

Generating white (Gaussian) noise on FPGA devices seems to be simple on the first glance. First, place and route the logic for the main application (that includes particularly a cryptographic component) on the FPGA device. Then, connect all yet unused (but still routable) resources of the device to some random data source and clock them accordingly to the chip enable signal of the cryptographic component. This can even be done automatically, i.e., is certainly possible to create a tool that detects and configures yet unused logic for noise generation in a subsequent development step.

However, our goal in this section is much more specific: we intend to investigate how to configure the available FPGA resources in a way that we achieve a maximum noise level. More precisely, we now analyze three power-consuming strategies that are based on cascading and misusing FPGA resources.

Shift Register LUTs (SRL) Toggling the level of an input signal is known to have the highest impact on a gate's power consumption in CMOS devices – what also holds for SRAM-based FPGA devices. Thus, to generate high noise, we need to toggle as many signals as possible. Taking a closer glance at modern

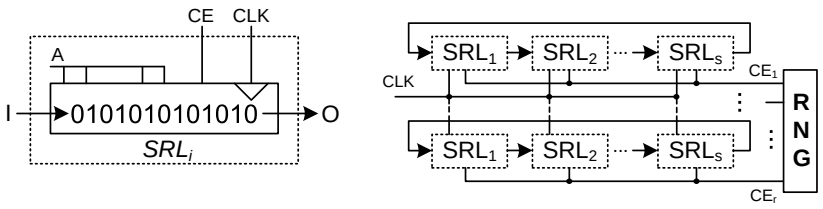


Fig. 1. Noise generation based on shift-register LUT (SRL)

FPGA architectures, these devices consist of large amounts of combined logic functions made up from flip-flops and lookup tables (LUT). LUTs with n inputs and m outputs can be configured as an n-to-m logic function generator (typically, $n = 4$ or $n = 6$ and $m = 1$). A straightforward approach would then cascade a large number of such LUT/flip-flop pairs (with the LUT being configured as logical NOT) and clock these elements according to a connected random source. However, we certainly could do better. Looking more into the details, a LUT itself consists of 2^n storage bits representing the truth table of its logic function. As a secondary function, the truth table of these LUTs can often be configured as 2^n-bit shift-register LUT (SRL) providing significant savings with respect to conventional shift registers made up from cascades of flip-flops. For an effective noise source, we now configure r cyclic rings of s SRL elements initialized with an alternating (toggle) bit pattern and connect the chip enable signal for each ring to r output bits of a random number generator[1]. The two parameters r and s control the amount of noise variance and noise amplitude, respectively. Figure 1 sketches the noise generating circuit using $r \times s$ SRL elements. Please note that this noise generator does not have an output, hence synthesis tools usually trim such unconnected components. Therefore, the `KEEP` attribute needs to be applied in the HDL description for such constructions to override an undesired optimization/removal by the tools.

BRAM Write Collisions (BWC) Another general observation for hardware devices is that irregular behavior often leads to increased power consumption. A write collision, for example, can occur in the dual-ported block memories (BRAM) of FPGAs when different data is written at the same memory address of a BRAM. For Xilinx FPGAs, for example, the result of such an incident is just undefined [26]. Indeed, it was shown in [9] that the different driving directions lead to data contention on the internal bus lines resulting in metastabilities within the inverter pair of a storage cell. We therefore assume the opposite and conflicting driving directions of the two memory ports to lead to an increased power consumption. We investigated and evaluated this effect using a construction with r BRAMs and s-bit port width as shown in Figure 2. Note that BRAMs are typically a scarce resource in most FPGA applications. But since we only

[1] For test implementations in this work, we used simple PRNGs, however the ideas can easily be combined with available TRNG constructions for FPGAs, e.g., see [25].

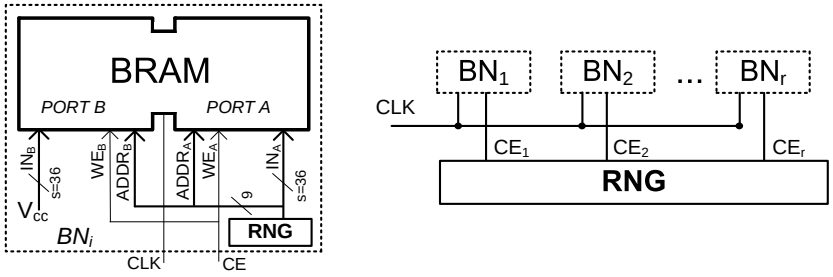


Fig. 2. Noise generator based on memory write collisions

need a single/few empty memory lines to create a write collision, we can also reuse (inactive) BRAMs for noise generation that are actually used otherwise in the main application and whose memory is not entirely used.

Short Circuits in Switch Boxes (SC) Producing a short circuit in a hardware circuit is certainly another strategy to significantly increase the power consumption of a device. Hence, we now try to generate a controlled short circuit (SC) on an FPGA for a very limited amount of time. Note that creating SCs in such a controlled way is not easy with FPGAs, since the design tools run sophisticated design rule checkers that inhibit any misconfiguration. Thus, the development of SC elements need to be done manually and cannot be simplified using HDL tools.

Moreover, we need to consider that SCs have the potential to damage a device. However, FPGA vendors are faced with this issue already by design. Recall that the vendors need to make sure that a configuration file cannot damage a device even if it is corrupted. In case such an illegal configuration is loaded into the FPGA in serial manner, many SCs can happen before the integrity check is finally able to detect the invalid device state. This takes place, in particular, on Xilinx Virtex devices with enabled bitstream encryption when an encrypted configuration is loaded for which no or an incorrect key is present (in this case, the FPGA is getting noticeably hot). Hence, FPGA vendors typically limit the strength of all conflicting drivers to less than 100 μA. Therefore, we can conclude that intentionally constructing short circuits should not have severe consequences on an FPGA's constitution or lifetime.

In order to create an SC in an FPGA configuration based on our Xilinx FPGA setup (cf. Section 3), we refer to the work by Beckhoff et al. [2] which lately demonstrated that LUT input multiplexers of the switch boxes are the most promising source to generate high-power SCs. We used Xilinx low-level tools to create three interconnected LUTs (i.e., two first-level LUTs sourcing a second-level LUT with two input multiplexers). Since it is not possible to directly define the state of routing input multiplexers, we employed Xilinx Design Language (XDL) to convert our placed design into a textual representation that then allows to modify all programmable interconnect points (PIP) freely. We manually

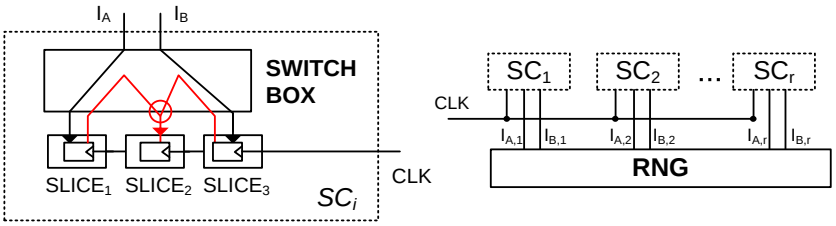


Fig. 3. Short circuits at the input multiplexer to a logic slice (denoted by red wires)

reconnected the outgoing PIPs of the first-level LUTs to the identical pinwire of the second-level LUT. This structure (see Figure 3) is then converted into a hard macro that can then be placed multiple times by black-box instantiation inside the FPGA configuration, providing a large number of controllable SC elements. Note that modern FPGAs contain several thousands of LUTs and corresponding input multiplexers. Hence we can easily insert r instances with nearly no resource overhead to scale the amount of noise accordingly to our needs. Note, however, that SC elements should always be spread among the entire chip to distribute the SC load to different power regions to avoid unintended side-effects.

2.2 Clock Randomization (CR)

DPA attacks need to exactly identify the point in time of a power trace when cryptographic data is processed. In order to complicate data alignment, randomized delays or dummy cycles are inserted into the cryptographic operation either by special state machines or non-deterministic processors [3, 10]. In this work, we present a novel and very efficient way to randomize data processing by using irregular clock cycle delays and multi-phase shifting obtained from digital clock managers (DCM) in FPGAs. Note that modern FPGAs usually contain a large number of DCMs (often $\geq$4) and clock buffers ($\geq$16) of which many remain unused in typical applications. Hence, the following proposal is very appealing to use this type of resource which would be wasted otherwise.

The integrated clock-management functions of many FPGA devices allow jitter correction, clock scaling and phase shifted clock signals. Clock buffers are placed on strategic places of the FPGA to optimize clock distribution. They also enable clock multiplexing (e.g., to drive the design temporarily at reduced clock frequency to implement processor sleep modes) that intrinsically provides a minimum cycle preservation. More precisely, assume we have two different clocks that are multiplexed via a clock buffer to drive a component of the FPGA design. When the clock input is requested to change from one to the other, the clock buffer will wait until the currently selected first clock is low/goes low and remains low until the second input clock has made a transition from high to low. After that, the second clock starts driving the output. In addition, the behavior can be interrupted (resulting in a wait state of undefined length) in case the clock

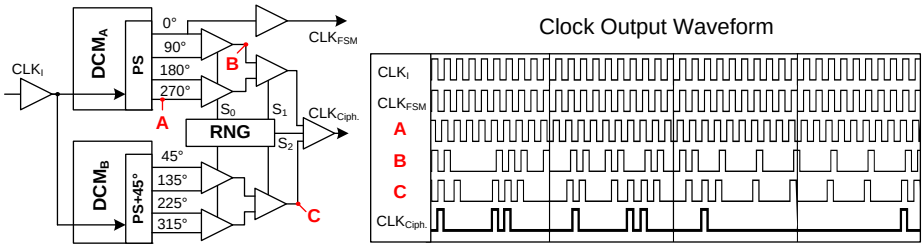


Fig. 4. Clock randomization using DCMs and a tree of clock buffers

multiplexer is requested to switch clocks again before the first clock change has been completely finished [26].

DCMs in Xilinx FPGAs directly provide outputs for clocks with fixed phase shifts of 0°, 90°, 180°and 270°. Furthermore, the phase of the output clock can also be set to a custom value (which can also be changed dynamically during runtime what is not considered in this work). Our clock randomization countermeasure makes use of a set of l DCMs providing n different output clocks, each phase-shifted by a fixed amount of $360/n$ degrees. A tree of $n-1$ clock multiplexers combines the different clocks to a single clock output that drives the cryptographic core. In addition to that, we need two further clock buffers to sample the input clock and to generate a system clock that is used for the remaining non-cryptographic part of the application (and for noise generating countermeasures). A sample design of this countermeasure with $n = 8$ different phase-shifted clocks is shown in Figure 4.

2.3 Preventing Clock Frequency Manipulations (PCM)

DPA attacks are usually performed at rather low frequencies to easily allow visual peak inspection of the power traces. However, DPA attacks are also at higher clock frequencies possible (e.g., at more than 100 MHz, see [17]), but become more complex due to low-filtering effects of the chip. Hence, to achieve a simple attack setup, an attacker usually desires to reduce the external clock frequency driving the FPGA. This can be prevented as follows. First, a DCM always requires a specific minimum input frequency (specified by the FPGA vendor), otherwise it may not lock (and the main application will not start). Second, a system designer can easily include a detector that triggers an alarm as soon as the clock falls below a specified minimum clock frequency. Figure 5 shows a suitable clock measurement circuit that uses a fixed path delay to shift the phase of a target clock by a fixed amount of $d = 180 + a$ where $a > 0$ denotes an additional phase margin to overcome clock jitter of the input clock. When an attacker attempts to reduce the external input clock frequency (or manipulate the duty cycle) beyond this margin, either one of the flip-flops will sample the alternate part of the clock period, causing finally the alarm to be triggered.

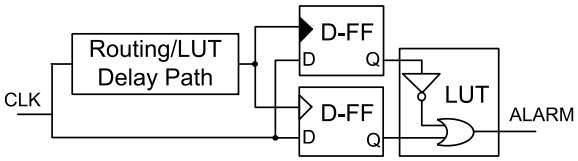


Fig. 5. Circuit to detect external clock manipulations

2.4 Block Memory Content Scrambling (BMS)

In this section we present a novel hardware countermeasure for FPGAs based on BRAM-based S-box/T-box scrambling. In many symmetric ciphers, S-boxes are used to introduce a non-linear component in the encryption process and are usually implemented as simple lookup tables. Depending on their size and construction, S-boxes can be realized either using (large amounts of) LUTs or block memories on an FPGA. DPA attacks typically focus on the input and/or outputs of the (known) S-boxes, hence a well-studied countermeasure attempts to mask the S-box data with readily changing, random values. However, it turned out that such a system either significantly reduces the encryption performance and/or requires costly additional operations to pass a random mask through the non-linear S-box [7, 8]. Similar to the concept of random permutation tables by Coron [4], we now build a freely running S-box masking scheme specifically for FPGA device. We first assume that we can rewrite the round function $y = R(x)$ of an arbitrary symmetric cipher as composition of a linear part $L(x)$ (including a linear key addition function) and a non-linear part $N(x)$ (implemented as a-on-b-bit S-box), resulting in $y = L(N(x))$ as round function.[2] Assume now N is realized as a lookup table using one port of a dual-ported BRAM of an FPGA device. We further assume that N occupies less than half of the memory available in the BRAM (i.e., 18KBit/36KBit for Xilinx devices). Now we define two memory segments or *contexts* in the BRAM: an active context which contains a recent version of the (masked) S-box currently used for encryption and an inactive context containing a copy which is currently scrambled and remasked by an encryption-independent process. The scrambling itself is a sequential process on the second port of that BRAM that applies a b-bit mask $\mathbf{m}$ to each S-box entry at address i of the active context and stores the result at address $i \oplus \pi(L(\mathbf{m}))$ of the inactive context (here, $\pi(x)$ represents a selection function for the corresponding S-box input bits if $a \neq b$). In other words, it applies an additive mask to the S-box output that is also pushed through the linear part L of the round function to determine the new (permuted) input address index

[2] Note that this generalization actually holds for a large number of ciphers, though symmetric round constructions often contain several linear operations $L_1, L_2, L_3, \ldots$ involving constructions such as $y = L_1(L_2(N(L_3(\ldots(x)))))$. But assuming that only a single non-linear component N is used in the round function, we can always combine and rearrange subsequent rounds in a way that the linear subcomponents aggregate into a single L as shown above and as could be seen in [4].

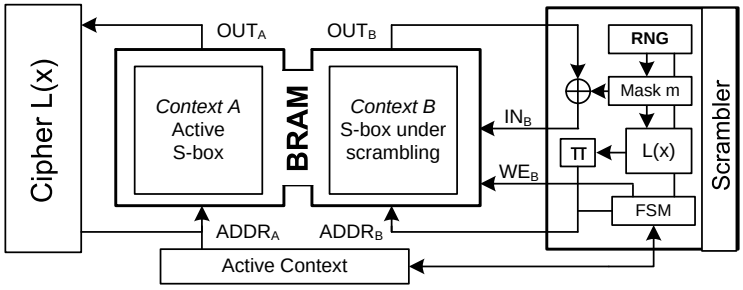


Fig. 6. Construction scrambling S-box entries with random masks

where each updated masked S-box value is finally stored. Note that we refer to this process (that adds a random b-bit mask $\mathbf{m}$ iteratively to all S-box entries and writes each entry back to a permuted address) as *scrambling* rather than *masking*. After all S-box entries have been processed by this scrambling process, the context can be switched (i.e., the inactive context becomes active and vice versa) and scrambling restarts on the recently deactivated context. Note that the cipher and scrambler operate concurrently, however, the context switch is never done within a running encryption to avoid data inconsistency issues. This implies that multiple sequential rounds and encryptions are performed using the same scrambled S-box and mask $\mathbf{m}$ (what could be exploited from a theoretical point of view using higher-order attacks). We like to stress at this point that we designed this countermeasure for performance and efficiency, since the concurrent data scrambling process and the instant context switch does not reduce the encryption speed. Note further, that in combination with previous countermeasures such as clock randomization and noise generation, higher-order attacks will also become extremely complex. Figure 6 shows finally the generic construction of the BRAM scrambling circuitry.

3 Case Study

We now investigate how the countermeasures presented above can harden an AES implementation against DPA attacks. We start with an unprotected AES instance which we subsequently augment with our countermeasures to evaluate their effectiveness.

3.1 Reference Architecture

For our experiments we used an unprotected, round-based T-table implementation of the standardized AES block cipher with a 128-bit data path. The round function for such an implementation uses four 8-to-32-bit T-tables T_i to compute a 32-bit share S_j of the 128-bit AES state S according to the following formula:

$$S_j = k_j \oplus T_0[\pi_{0,j}(S)] \oplus T_1[\pi_{1,j}(S)] \oplus T_2[\pi_{2,j}(S)] \oplus T_3[\pi_{3,j}(S)], \qquad (1)$$

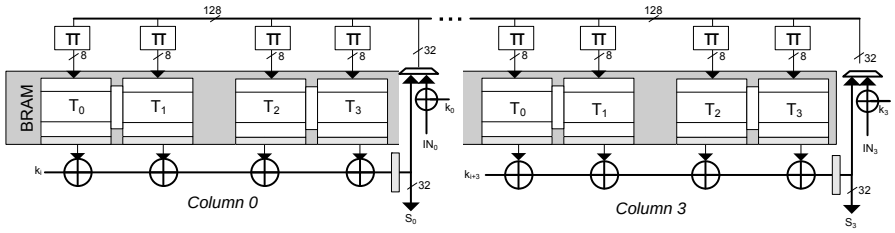


Fig. 7. T-table AES implementation used for this case study

where $\pi_{i,j}$ represents a static input byte selection function. Note that we can rewrite Equation (1) to

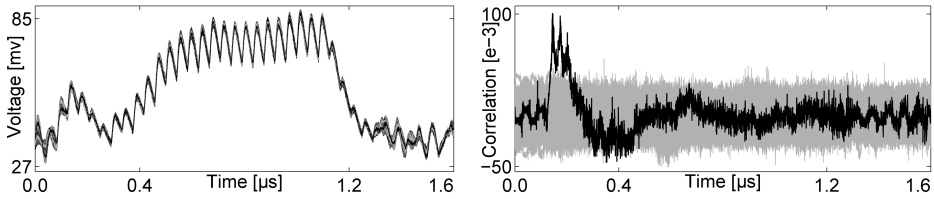
$$R_j(S) = L(N_0(\pi_{0,j}(S)), N_1(\pi_{1,j}(S)), N_2(\pi_{2,j}(S)), N_3(\pi_{3,j}(S)), k_j) \qquad (2)$$

with $N_i = T_i$ and $L(a, b, c, d, e) = a \oplus b \oplus c \oplus d \oplus e$ to comply with the generic round function specification discussed in Section 2.4. Further information about the AES T-table implementations can be found in [6]. Our unprotected AES implementation as shown in Figure 7 requires 21 clock cycles (1 initial clock cycle and 2 for each round) to compute a full AES-128 encryption and consumes 682 slices (1182 LUTs, 397 FF) and 8 BRAMs storing the 16 T-tables on a Xilinx Virtex-II Pro FPGA.

3.2 Measurement Setup and Attack Model

The AES design explained above was implemented on the Xilinx Virtex-II Pro FPGA (xc3vp7) of a SASEBO circuit board which is particularly designed for side-channel attack evaluations [1]. The instantaneous power consumption traces are collected using a LeCroy WP715Zi 1.5 GHz oscilloscope at a sampling rate of 2.5GS/s and by means of a differential probe capturing the voltage drop across a 1Ω resistor placed in the VCCINT (1.6V) path of the target FPGA.

In order to examine the leakage of the implementation and find a suitable power model for the correlation power analysis (CPA) attacks, we started the practical experiments when the target core is clocked at 24MHz which is selected as the reference implementation for further comparisons. Figure 8(a) shows a superposition of 1000 traces of this case indicating the well alignment of the measurements. Since the I/O ports of the BRAMs of the target FPGA have a considerably higher capacitance compared to the other low amount of logic cells of our target design, data transfered through the BRAM buses caused by reading the memory cells while computing the T-table outputs should have noticeable impact on the power consumption. Therefore, we have used the Hamming weight (HW) of the 32-bit T-table output as the hypothetical power model in a CPA attack. The result of such an attack predicting the T-table outputs of the first encryption round using 10 000 traces is shown in Figure 8(b). It shows that – using a *rule of thumb* [17] – around 3000 traces are the minimum number



(a) (b)

Fig. 8. The AES core in 24MHz (a) superimposition of 1000 traces, (b) CPA attack result using 10 000 traces

of required measurements. We have examined a couple of other hypothetical models, and the best result was achieved using the aforementioned model.

4 Evaluation and Results

The results of the attacks evaluating the effect of each method introduced in Section 2 are presented in the following. Since the leakage model of the target device is well known and can be appropriately estimated by a HW model, we limited our evaluations to CPA attacks and considered the number of required measurements as metric for comparisons. The result of each scheme is compared to the results shown above in the reference implementation.

4.1 Noise Generators

Adding each of the noise generation schemes individually as explained in Section 2.1 leads to an increased amount of switching noise and has an effect on the number of required measurements for a successful attack. In order to practically examine their effectiveness, we added the noise generators based on shift-register LUTs (SRL), BRAM write collisions (BWC) and short circuits (SC) individually into the reference implementation and repeated the measurements and corresponding attacks. We used uniform parameters for the individual noise generators, namely $r = 16$ and $s = 36$, taking an additional resource consumption of 576 LUTs (SRL), 16 BRAMs (BWC) and 48 LUTs (SC), respectively. Figure 9 shows the result of the attacks when each of the noise sources is separately enabled in addition to the case when all of them exist in the design. It can be concluded that adding noise sources with quite moderate parameters already increases the number of required measurements slightly, i.e., to around 8000. In this context, the SC noise generator is obviously the most efficient one with respect to the number of consumed resources. However, using solely noise addition (even with much larger parameters for r and s) can certainly not be considered as an optimum way to make any attacks infeasible.

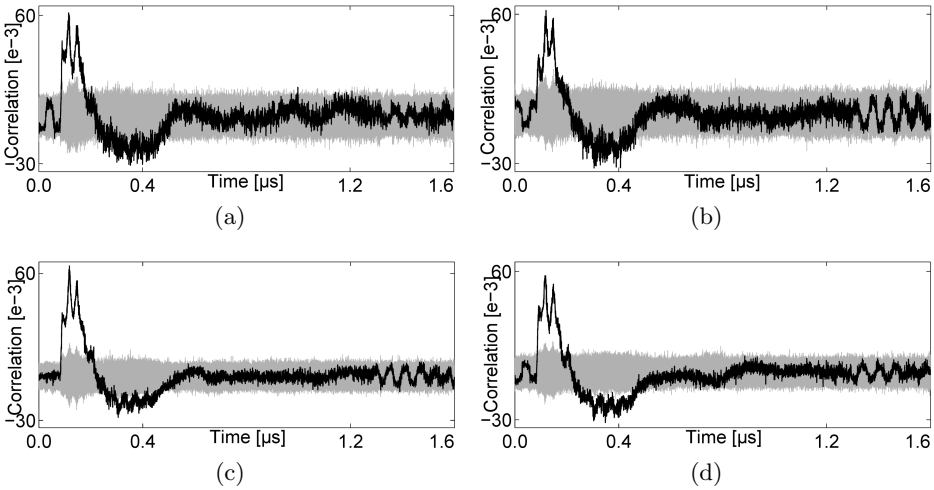


Fig. 9. CPA attack results using 50 000 traces of the AES core in 24MHz including (a) shift-register LUTs, (b) BRAM write collisions, (c) short circuits, and (d) all three noise sources

4.2 Clock Randomizing

In Section 2.2 we presented the CR method to randomize the clock source by randomly changing the clock phase to introduce a variable misalignment of the power traces. For our attack, we used a setup based on $l = 2$ DCMs with $n = 8$ phase-shifted clock outputs which are processed and multiplexed by 9 clock buffers. An encryption clocked by this irregular output takes on average 3.77 times longer than our reference implementation. Embedding this unit into the reference implementation – as expected – led to a variable amount of time required for an encryption. Figure 10(a) shows a superimposition of 1000 traces, already indicating a strong misalignment. Therefore, we performed the attacks using considerably more measurements, i.e., 10 000 000. The results shown in Figure 10(b) signifies the need for around 3 000 000 traces to determine the correct key hypothesis. We like to emphasize that the randomization of processing times is not aligned with the primary clock any longer (unlike in shuffling schemes), rendering a combing technique [24] useless in our case. Since combing is done by adding up the leakage points of consecutive clock cycles of a trace while – as shown in Figure 10(a) – it is here not possible to clearly distinguish the clock cycles. Reducing the input clock frequency to facilitate the attack can be prevented by using a detector for clock manipulations as shown in Section 2.3. However, a windowing approach [3], summing up all points within a defined window, proved to be effective since the operating clock frequency of 24MHz let the power peaks of consecutive clock cycles overlap with each other (i.e., this intrinsic low pass filter has the same effect as windowing). Indeed, we even repeated

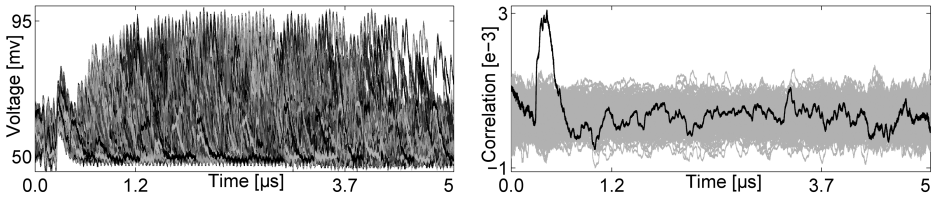


Fig. 10. The AES core in 24MHz equipped with the clock phase shift unit (a) superimposition of 1000 traces, (b) CPA attack result using 10 000 000 traces

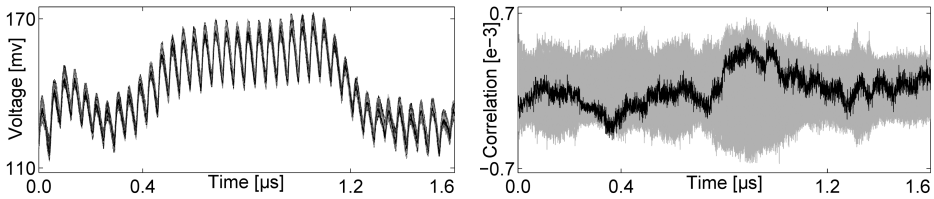


Fig. 11. The AES core in 24MHz equipped with BRAM scrambling (a) superimposition of 1000 traces, (b) CPA attack result using 100 000 000 traces

this type of attack with different window sizes, but with no significant difference to Figure 10(b).

4.3 Block Memory Content Scrambling

In order to integrate the BRAM scrambling technique (BMS) introduced in Section 2.4 into our reference implementation, we need to duplicate the number of BRAMs to provide a separate memory port and space for the scrambling process. For the scrambling, we used 16×32 bit masks $\mathbf{m}_i$ to mask the output of all 16 T-tables requiring a total of 512 random bits per scrambling cycle. According to Equation (2), we can apply the linear transformation L of the 32-bit AES round function to the masks by computing $L(\mathbf{m}_i, \mathbf{m}_{i+1}, \mathbf{m}_{i+2}, \mathbf{m}_{i+3}, 0) = \mathbf{M}_i$ for $i = 0 \ldots 3$. From each aggregated 32-bit mask $\mathbf{M}_i$ the corresponding input byte mask is finally selected by $\pi(\mathbf{M}_i)$ and determines the corresponding permutation mask to the next round's T-table input.

Since this scheme does not affect the timing behavior of the design, the variety of power traces should be similar to that of the reference implementation. One difference that may have an influence on the power consumption is the additional scrambler circuit that concurrently modifies the BRAM contents. Therefore, as shown in Figure 11(a) the DC level of the power traces is increased compared to Figure 8(a).

The input and output of the T-tables are now masked by random values which are not shared inside a single round computation, and we expect this design to

avoid univariate power analysis attacks. In order to examine this claim, we have measured 100 000 000 traces and have performed the same attack as before. The result is shown in Figure 11(b).

However, a critical point is the reuse of the masks between the subsequent rounds. This means, the 512-bit mask used in the first round is reused in the later rounds since the BRAM sections used during an encryption process stay unaltered. Therefore, a second-order attack is possible combining the leakages, for example, of two consecutive rounds. To do this, the adversary needs to predict at least 2^{40} bits of the key, i.e., 2^{32} for 4 bytes of the first round key which are at the same column after ShiftRows, and 2^8 for one byte of the second round key. One may also think about combining the leakages of the scrambler module with that of the first encryption round, but this requires to know exactly the instance in time when the scrambler processes the T-table content used in the next encryption. Note further that the scrambler unit operates independently with a separate clock, hence its computations are not synchronized with the encryption process.

4.4 Combining Countermeasures

Although we were already unable to defeat the latter countermeasure with first-order attacks and 100 million traces, we like to stress that we still can strengthen this design by adding the clock randomization and noise generation countermeasures. We have omitted to provide it as a separate result since it is similar to the one depicted in Figure 11(b). The combination of all proposed countermeasures are likely to harden our design against a large number of multivariate attacks which are out of scope of this work. The designer thus can mix and match the proposed countermeasures according to the security requirements of the application and the remaining logic available on the FPGA device. Table 1 shows the overhead of the countermeasures for the generic and specific case considering their impact on the logic resource consumption and execution time of a cryptographic process. Comparing our results with other work, we noticed that countermeasures specifically built for (comparable) FPGA devices are quite rare. We thus only refer to the work [18] implementing an SCA countermeasure based dynamic partial reconfiguration on a Virtex-II Pro. Their countermeasure requires an overhead of 2566 slices (i.e., up to 5132 LUT/FF pairs) and reduces the throughput of a plain AES implementation by a factor of 6.6. Except for BWC, combining all our countermeasures of this work results in a more efficient solution (i.e., a combination of all methods, excluding BWC, consumes 2337 LUT, 1171 FF, 1 DCM, 7 CB and reduces the throughput by a factor of 3.77).

5 Conclusion

In this work we presented several generic hardware countermeasures against DPA attacks that can be efficiently implemented on recent FPGA devices. We could practically demonstrate that a combination of the presented countermeasures

Table 1. Time and resource overhead for each proposed countermeasure for parameters $r = 16$, $s = 36$ and $l = 2$, excluding RNG and output combining function. The plain AES implementation consumes 8 BRAM, 1182 LUT and 397 FF on a Xilinx Virtex-II Pro. Figures with asterisk (*) are not quantified due to implementation-specific characteristics. Abbreviations: LUT=Look-Up-Table, FF=Flip-Flop, DCM=Digital Clock Manager, BRAM=Block Memory, CB=Clock Buffer.

Section/ Method	**Generic Implementation** *Logic*	*Time*	**AES T-Table Case Study** *Logic*	*Time*
2.1.1/SRL	$r \cdot s$ LUT	*none*	576 LUT	*none*
2.1.2/BWC	r BRAM, $r \cdot s$ LUT	*none*	16 BRAM, 576 LUT	*none*
2.1.3/SC	$3 \cdot r$ LUT	*none*	48 LUT	*none*
2.2/CR	$l - 1$ DCM, $4l - 1$ CB	*n/a**	1 DCM, 7 CB	3.77×
2.3/PCM	1 LUT, 2 FF, *Delay Path*	*none*	3 LUT, 2 FF	*none*
2.4/BMS	*n/a**	*none*	8 BRAM, 1706 LUT, 1169 FF	*none*

(noise generation, clock randomization and memory scrambling) rendered first-order DPA attacks with up to 100 million traces unsuccessful. For independent verification of our results and future work (e.g., second-order SCA), the PROM files with the AES implementation and all countermeasures for the SASEBO are publicly available at `http://www.emsec.rub.de/research/publications`.

References

1. Side-channel Attack Standard Evaluation Board (SASEBO). Further information are available via `http://www.rcis.aist.go.jp/special/SASEBO/index-en.html`.
2. C. Beckhoff, D. Koch, and J. Torresen. Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration. In *FPL*, pages 596–601. IEEE Computer Society, 2010.
3. C. Clavier, J.-S. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya. Koç and C. Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 252–263. Springer, Aug. 2000.
4. J.-S. Coron. A new DPA countermeasure based on permutation tables. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN 08: 6th International Conference on Security in Communication Networks*, volume 5229 of *LNCS*, pages 278–292. Springer, Sept. 2008.
5. J.-S. Coron and I. Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 95–109. Springer, Aug. 2010.
6. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
7. J. D. Golic and C. Tymen. Multiplicative masking and power analysis of AES. In B. S. Kaliski Jr., Çetin Kaya. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 198–212. Springer, Aug. 2002.
8. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya. Koç and C. Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Aug. 1999.

9. T. Güneysu. Using Data Contention in Dual-ported Memories for Security Applications. *Journal of Signal Processing Systems*, pages 1–15, 2010.
10. J. Irwin, D. Page, and N. P. Smart. Instruction Stream Mutation for Non-Deterministic Processors. In *ASAP*, pages 286–295. IEEE Computer Society, 2002.
11. K. Itoh, J. Yajima, M. Takenaka, and N. Torii. DPA countermeasures by improving the window method. In B. S. Kaliski Jr., Çetin Kaya. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 303–317. Springer, Aug. 2002.
12. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Aug. 1996.
13. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Aug. 1999.
14. F. Macé, F.-X. Standaert, and J.-J. Quisquater. Information theoretic evaluation of side-channel resistant logic styles. In P. Paillier and I. Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 427–442. Springer, Sept. 2007.
15. H. Mamiya, A. Miyaji, and H. Morimoto. Efficient countermeasures against RPA, DPA, and SPA. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 343–356. Springer, Aug. 2004.
16. S. Mangard. Hardware countermeasures against DPA? a statistical analysis of their effectiveness. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *LNCS*, pages 222–235. Springer, Feb. 2004.
17. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
18. N. Mentens, B. Gierlichs, and I. Verbauwhede. Power and fault analysis resistance in hardware through dynamic reconfiguration. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 346–362. Springer, Aug. 2008.
19. A. Moradi and A. Poschmann. Lightweight cryptography and DPA countermeasures: A survey. In R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, editors, *FC 2010 Workshops*, volume 6054 of *LNCS*, pages 68–79. Springer, Jan. 2010.
20. K. Okeya and T. Takagi. A more flexible countermeasure against side channel attacks using window method. In C. D. Walter, Çetin Kaya. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 397–410. Springer, Sept. 2003.
21. E. Prouff and R. P. McEvoy. First-order side-channel attacks on the permutation tables countermeasure. In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 81–96. Springer, Sept. 2009.
22. F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In A. Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, Apr. 2009.
23. F.-X. Standaert, S. B. Örs, and B. Preneel. Power analysis of an FPGA: Implementation of Rijndael: Is pipelining a DPA countermeasure? In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 30–44. Springer, Aug. 2004.
24. S. Tillich and C. Herbst. Attacking state-of-the-art software countermeasures-a case study for AES. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 228–243. Springer, Aug. 2008.
25. M. Varchola. *FPGA Based True Random Number Generators for Embedded Cryptographic Applications*. PhD thesis, Technical University of Kosice, 2008.
26. Xilinx Inc. User Guides for Xilinx FPGA devices, April 2011. `http://www.xilinx.com`.