On 3-share threshold implementations for 4-Bit S-boxes

Kutzner, Sebastian; Nguyen, Phuong Ha; Poschmann, Axel; Wang, Huaxiong

2013

Kutzner, S., Nguyen, P. H., Poschmann, A., & Wang, H. (2013). On 3-Share Threshold Implementations for 4-Bit S-boxes. 4th International Workshop, Constructive Side-Channel Analysis and Secure Design (COSADE) 2013, 7864, pp.99-113.

<https://hdl.handle.net/10356/99651>

https://doi.org/10.1007/978-3-642-40026-1_7

© 2013 Springer-Verlag Berlin Heidelberg. This is the author created version of a work that has been peer reviewed and accepted for publication by 4th International Workshop, Constructive Side-Channel Analysis and Secure Design (COSADE) 2013, Springer-Verlag Berlin Heidelberg. 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:
[http://dx.doi.org/10.1007/978-3-642-40026-1_7].

Downloaded on 26 Apr 2023 18:43:39 SGT

On 3-share Threshold Implementations for 4-bit S-boxes

Sebastian Kutzner^{1,2}, Phuong Ha Nguyen^{1,2}
Axel Poschmann^{1,2} and Huaxiong Wang²

¹ PACE Temasek Laboratories,
² Division of Mathematical Sciences, SPMS,
Nanyang Technological University, Singapore
{skutzner, phuongha, aposchmann, hxwang}@ntu.edu.sg

Abstract. One of the most promising lightweight hardware countermeasures against SCA attacks is the so-called Threshold Implementation (TI) [12] countermeasure. In this work we discuss issues towards its applicability and introduce solutions to boost its implementation efficiency. In particular, our contribution is three-fold: first we introduce two methodologies to efficiently implement 3-share TI to a given S-box. Second, as an example, we successfully apply these methodologies to PRESENT and are able to decrease the area requirements of its protected S-box by 37-40%. Third, we present the first successful practical Mutual Information Attack on the original 3-share TI implementation of PRESENT and compare it with a correlation-enhanced collision attack using second-order moments.

1 Introduction

Side Channel Attacks (SCA) [6] were introduced in 1997 by Kocher *et al.* and exploit the fact that while a device is processing data, information about this data is leaked through different channels, e.g., power consumption, electromagnetic emanation and so forth. DPA [7] is a well known technique analyzing multiple measurements with statistical methods. It exploits the correlation between intermediate results, which partly depend on a secret, and the power consumption.

Several countermeasures (far too many to address all of them) have been proposed during the last years, for example, to decrease the SNR ratio [9], to balance the leakage of different values [13] or to break the link between the processed data and the secret, i.e., masking [9]. Due to the presence of glitches, masked implementation might still be vulnerable to DPA [9]. A recent countermeasure against DPA was introduced in 2006 by Nikova *et.al.* [11] and is called *Threshold Implementation* (TI). It is based on secret sharing (or multi-party computation) techniques and is provable secure against first order DPA even in the presence of glitches. Furthermore, it can be implemented very efficiently in hardware [14].

The number of shares required for a TI depends on the degree d of the non-linear function (S-box) and [11,12] have shown that one needs at least $d + 1$ shares. It implies that the higher the degree of the non-linear function, the more

shares are required and the larger the implementation gets. Since a degree of two is the minimal degree of a non-linear function, the optimal number of shares is three. Therefore, to apply a 3-share TI to a larger degree function, this function must be represented as a composition of quadratic functions [14].

In this work we introduce two methodologies to efficiently implement 3-share the TI countermeasure to an S-box (Section 2). We then successfully apply these methodologies to the PRESENT S-box, reducing the hardware implementation costs of the S-box layer by 37 – 40% (Section 3). Last, we investigate the security of our new design with practical measurements. In addition, we present a successful practical Mutual Information attack against the original TI implementation of [14] and compare it with a correlation-enhanced collision attack using second-order moments [10] (Section 4). Finally, the paper is concluded in Section 5.

2 One S-box for all

In this section, we introduce two methodologies to improve the hardware implementation costs for TI. To illustrate our contribution we chose PRESENT as an example. Figure 1 shows how to apply the TI to a 4-bit S-box: first it is decomposed into two stages $G(\cdot)$ and $F(\cdot)$ (horizontal), then each stage is shared (vertical).

Figure 1 also shows that in [14] the authors implemented $F(\cdot)$ and $G(\cdot)$ using *six different* 8×4 vectorial Boolean functions $f_1, f_2, \dots, g_3$. In the following we will show how to implement the same functionality with only *one* 8×4 vectorial Boolean function, which significantly reduces the area/memory requirement of the TI S-box layer.

Fig. 1. Decomposition of an S-box [14]

According to [1], only the 4-bit S-boxes in alternating group A_{16} can be decomposed into quadratic permutations. The interested reader is referred to [1] for more details. [8] proposed a classification of 4-bit S-boxes based on the following relationship:

Definition 1. [8] Two S-boxes $S(x), S'(x)$ are linearly equivalent if and only if there exist two 4×4 -bit invertible matrices A, B and two 4-bit vectors c, d such that

$$S'(x) = A(S(Bx \oplus c) \oplus d), \forall x \in \{0, \dots, 15\}$$

The definition above shows us that we can convert one S-box to another from the same class simply by using linear transformations. We will use this fact later for optimizing hardware implementations of a given S-box.

2.1 The horizontal level

In order to apply the 3-share TI to a cubic S-box $S(\cdot)$, first the S-box is decomposed into a composition of two quadratic permutations $F(\cdot)$ and $G(\cdot)$ (see Figure 1).

Observation 1 Assume a vectorial Boolean function $S(\cdot) = G(G(\cdot))$, where $G(\cdot)$ is a vectorial boolean function. Then the hardware implementation costs of $S(\cdot)$ may be reduced by reusing the implementation of $G(\cdot)$. Experiments have shown that the costs for additional logic, e.g., a multiplexer, is less than implementing $G(\cdot)$ twice. Numbers are provided in Section 3.

The main problem of Observation 1 is how to find a suitable $G(\cdot)$ such that $G(G(\cdot))$ belongs to the desired class, e.g., class C_{266}^4 for the PRESENT S-box [1]. We discovered that the only classes reachable by the construction $G(G(\cdot))$ are $C_{150}^4, C_{151}^4, C_{168}^4, C_{214}^4, C_{215}^4, C_{233}^4, C_{234}^4, C_{236}^4, C_{238}^4, C_{243}^4, C_{244}^4, C_{259}^4, C_{260}^4, C_{264}^4, C_{266}^4, C_{286}^4, C_{292}^4, C_{296}^4, C_{297}^4$ from Table 12 in [1].

[14] states that there are around 2^{26} quadratic 4-bit permutations and they also propose a weighting function to estimate their hardware implementation costs. We used a computer program to select those quadratic permutations, which have smaller hardware implementation costs than the F, G used in [14]. We then checked to which class their composition belongs. and, finally, we selected the one combination of A, B, c, d, G with the smallest hardware implementation. This way we found the following quadratic $G(\cdot)$ such that $S'(\cdot) = G(G(\cdot))$ belongs to class C_{266}^4 :

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G(x)	0	4	1	5	2	F	B	6	8	C	9	D	E	3	7	A
G(G(x))	0	2	4	F	1	A	D	B	8	E	C	3	7	5	6	9

Let $G(x, y, z, w) = (g_3, g_2, g_1, g_0)$, where x denotes the most significant bit and w the least significant bit of the input, and g_3 denotes the most significant bit and g_0 the least significant bit of the output. Then its *Algebraic Normal Form* (ANF) is as follows: $g_3 = x + yz + yw$, $g_2 = w + xy$, $g_1 = y$ and $g_0 = z + yw$.

Using Definition 1 we know that the S-box of PRESENT $S(\cdot)$ is linearly equivalent to the found $S'(\cdot) = G(G(\cdot))$, i.e

$$S(x) = A(S'(Bx \oplus c) \oplus d) = A(G(G(Bx \oplus c)) \oplus d), \forall x \in \{0, \dots, 15\}.$$

It can be constructed with the following 4×4 -bit matrices A , B and 4-bit constants c , d :

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, c = (0001)_2 = 1, d = (0101)_2 = 5.$$

Generally, the quadratic permutation G should be chosen as follows: 1) its $G(G(\cdot))$ must be in the same linearly equivalent class as the given S-box S ; then the minimal hardware implementation for A , B , c , d and G should be selected.

2.2 The vertical level

In the second step $G(\cdot)$ has to be divided into three 8×4 vectorial Boolean functions $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$. In practice, all these vectorial Boolean functions are implemented separately. We can reduce the implementation costs by using the following observation:

Observation 2 *The vectorial Boolean functions of $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$ are equal, except for the indices of the inputs and the existence of constants.*

The observation is derived from the construction of the vectorial Boolean functions $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$. Again, the left most bit of G_i represents the most significant bit and the right most bit represents the least significant bit, respectively. If we take the latter constructed $G(\cdot)$, then:

$$G_1(x_2, y_2, z_2, w_2, x_3, y_3, z_3, w_3) = (g_{13}, g_{12}, g_{11}, g_{10})$$

$$g_{13} = x_2 + y_2 z_2 + y_2 z_3 + y_3 z_2 + y_2 w_2 + y_2 w_3 + y_3 w_2$$

$$g_{12} = w_2 + x_2 y_2 + x_2 y_3 + x_3 y_2$$

$$g_{11} = y_2$$

$$g_{10} = z_2 + y_2 w_2 + y_2 w_3 + y_3 w_2$$

$$G_2(x_1, y_1, z_1, w_1, x_3, y_3, z_3, w_3) = (g_{23}, g_{22}, g_{21}, g_{20})$$

$$g_{23} = x_3 + y_3 z_3 + y_1 z_3 + y_3 z_1 + y_3 w_3 + y_1 w_3 + y_3 w_1$$

$$g_{22} = w_3 + x_3 y_3 + x_1 y_3 + x_3 y_1$$

$$g_{21} = y_3$$

$$g_{20} = z_3 + y_3 w_3 + y_1 w_3 + y_3 w_1$$

$$G_3(x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2) = (g_{33}, g_{32}, g_{31}, g_{30})$$

$$g_{33} = x_1 + y_1 z_1 + y_1 z_2 + y_2 z_1 + y_1 w_1 + y_1 w_2 + y_2 w_1$$

$$g_{32} = w_1 + x_1 y_1 + x_1 y_2 + x_2 y_1$$

$$g_{31} = y_1$$

$$g_{30} = z_1 + y_1 w_1 + y_1 w_2 + y_2 w_1$$

Therefore, we only need to implement $G_1(\cdot)$ and then reuse it for $G_2(\cdot)$ and $G_3(\cdot)$ by re-arranging the inputs accordingly. Please note that this observation can also be applied to ANFs containing constants, e.g., the ANFs of the G function specified in the appendix of [14]. In those cases, it is easy to see, that constants can be added to those two shares which initially did not contain a constant to yield three equal vectorial Boolean functions.

Note 1. $G(\cdot)$ is a 4-bit permutation and its 3-shared version, which is the 12-bit vector $(g_{10}, g_{11}, g_{12}, g_{13}, g_{20}, g_{21}, g_{22}, g_{23}, g_{30}, g_{31}, g_{32}, g_{33})$, is a 12-bit permutation. Therefore, the uniformity property is satisfied.

Assume that the given function $G(\cdot)$ is protected by using a k -share TI and $G_1(\cdot), \dots, G_k(\cdot)$ are its shared functions. Based on the same observation given above, the hardware implementation of $G_i(\cdot), 1 \leq i \leq k$, can be reused. It should be noted that the shared version of $G(\cdot) = (G_1(\cdot), \dots, G_k(\cdot))$ may not satisfy the uniformity property, e.g., $(G_1(\cdot), G_2(\cdot), G_3(\cdot)) = (g_{10}, g_{11}, g_{12}, g_{13}, g_{20}, g_{21}, g_{22}, g_{23}, g_{30}, g_{31}, g_{32}, g_{33})$ may not be a 12-bit permutation. In this case, we have to use the re-masking method [12] to fulfill this property. Therefore, our technique can be applied not only for this special case but also in general whenever a function is shared.

3 Hardware implementation

In this section we will describe hardware implementations of PRESENT protected with the TI countermeasure with a shared data path and an unshared key schedule that is similar to profile 2 described in [14]. This profile is chosen because our approach allows dramatic area savings –up to 57%– within the **S-box** module at the expense of a slight area increase in the storage modules. Thus to allow a fair assessment of the overall area savings, we did not focus on the **S-box** module, but instead implemented a whole PRESENT. Note that we deliberately chose to not share the key schedule for two reasons: first the S-box used would be the same, and second, to focus on the main contribution of this paper – a new methodology to reduce the area cost of a 4-bit S-box. First we introduce the design flow used before we detail the hardware architectures and finally summarize the implementation results.

3.1 Design flow

For the hardware implementation in *VHDL*, we used the Boolean minimization tool **BOOM II** [3,2] to obtain the four ANFs of G . For functional simulation we used *Mentor Graphics ModelSimXE 6.4b* and *Synopsys DesignCompiler* version *E-2010.12-SP2* was used to synthesize the designs to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18μm 1P6M* logic process and has a typical voltage of 1.8 Volt [5].

For synthesis we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz.

3.2 Serialized architecture

Figure 2 depicts our architecture. The main differences between our design and profile 2 in [14] are the **S-box** module and a part of the storage modules for the shared data path. The three shares of the data path are stored in three identical replications of the storage module denoted by **State**, m_{d1} and m_{d2} . Each of them comprises of 60 flip-flops that can act as a normal 60-bit wide register (vertical shifting direction) or as a 4-bit wide 15 stages shift register (horizontal). The remaining 4 bits are stored in a similar way (denoted with I, II and III in Fig. 2) but with two additional 2-to-1 input MUXes (one for each shifting direction). Those 4 bits act as a shift register in a vertical way, allowing to change the input to G . The parallel 60-bit wide output is concatenated with the output of the 4-bit wide register and is transformed by the P-layer of PRESENT. The **Key** module stores the key state and performs the PRESENT key schedule.

Fig. 2. Architecture of a serialized TI-PRESENT-80 using our new optimization techniques.

The **S-box** module comprises of only *one* 8×4 vectorial Boolean function G (47 GE) that is used for all three shares and for both stages instead of *six* as used in [14]. Recall, that we implement the PRESENT S-box as $S(x) = A(G(G(Bx \oplus c)) \oplus d)$. Therefore, the inputs to G (II and III) are transformed by $Bx+c$ (two times 7 GE) and its output is temporarily stored for two clock cycles in two consecutive 4-bit flip-flops (48 GE). In order to process all shares re-using the same combinational logic, I, II and III are rotated vertically. Suppose the content of I, II and III are denoted a , b , and c , respectively, and the content of the 4 MSB of **Key** is denoted k . Then in the first clock cycle (b, c) are

processed yielding t_1 , in the second $(a \oplus k, b)$ yielding t_2 , and finally $(c, a \oplus k)$ yielding t_3 . Since, for the second stage, we do not need to process the input to G by $\mathbf{Bx} + \mathbf{c}$, we transform all three intermediate results t_1, t_2, t_3 by $\mathbf{B}^{-1}(\mathbf{x} + \mathbf{c})$ (21 GE)¹ and store the results G_1, G_2, G_3 in I, II and III. The aforementioned procedure –computing t_1, t_2, t_3 by rotating I, II and III– repeats for another three clock cycles, after which the second stage is completed. This time the three intermediate results t_1, t_2, t_3 are transformed by $\mathbf{Ax} + \mathbf{d}$ (18 GE) and stored in the shift registers **State**, $\mathbf{m}_{d1}$ and $\mathbf{m}_{d2}$, which are shifting horizontally, and the next 4-bit nibbles are ready to be processed.

The FSM module comprises of one initial state, six states for the S-box, one state for the permutation layer that is used instead of the sixth S-box state at the end of each round, a finished state that sets the done signal to high, and a done state. The output is gated by an AND-gate that only lets data pass to the final output XOR after 31 rounds have been processed.

It takes in total $6 * 16 = 96$ clock cycles for one round, hence the output is ready after 2976 clock cycles. During the 16 clock cycles required to output the result nibble-wise, the next message and key can be loaded, which takes 20 clock cycles. Thus in total our architecture requires 2996 clock cycles to process one message, compared to 578 clock cycles reported in [14], which is an overhead of a factor of around 5.2.

3.3 Round-based architecture

Based on the serialized implementation we estimate figures for round-based TI PRESENT-80 implementations using the normal TI S-boxes as used in [14] and using our new approach (see Table 1). The top-level glue logic consists mostly of XOR and AND gates that scale linearly with the data path width, hence we multiplied it by 16. All storage modules (**Key**, **State**, $\mathbf{m}_{d1}$, $\mathbf{m}_{d2}$) save around 9 GE for a 4-bit MUX that is only required for serialized implementations. The FSM now only consists of a round counter and some simple combinational logic, which we estimate to be around 50 GE. Our approach requires slightly more complex control logic, so we add 50% to be on the safe side. Compared to the normal TI implementation we require two additional 64-bit MUXes (298 GE) for each of the three shares, to allow the shares to be horizontally rotated. The major area increment occurs in the **S-boxes** module, which consists of 16 S-boxes. Due to the pipelining stage in its **S-box** module, the normal TI based implementation requires 2 clock cycles per round, so in total $2 * 31 = 62$ clock cycles. Our approach would require in total $6 * 31 = 186$ clock cycles, which is a timing overhead of only a factor 3.

3.4 Discussion

Our main goal is to investigate the savings that one can achieve using our new optimization technique, hence we compare our core to profile 2 as published

¹ Compared to using two MUXes (19 GE), this approach has the advantage of a simpler control logic at roughly the same area requirements.

Table 1. Breakdown comparison of the post-synthesis implementation results of a serialized PRESENT-80 are shown in the upper half and estimates for a round-based implementation are shown in the lower half. For each architecture two different memory strategies are shown: D-flip-flops with enable (D-FF + en), and scan-flip-flops with clock gating (s-FF + cg). All figures are Gate Equivalents (GE) and estimated figures are indicated with an *.

<i>Serialized Architecture</i>									
	Arch.	Etc.	Key	FSM	State	m_{d1}	m_{d2}	S-box	Sum
D-FF+en	[14]	58	778	139	587	587	587	351	3087
	<i>this work</i>	58	778	146	608	608	608	151	2957
	<i>Difference</i>	0	0	+7	+21	+21	+21	-200	-130
s-FF+cg	[14]	58	520	139	389	389	389	351	2235
	<i>this work*</i>	58	520	146	410	410	410	151	2105
	<i>Difference</i>	0	0	+7	+21	+21	+21	-200	-130
<i>Round-based Architecture</i>									
	Arch.	Etc.	Key	FSM	State	m_{d1}	m_{d2}	S-box	Sum
D-FF+en	<i>based on [14]*</i>	928	769	50	576	576	576	5616	9091
	<i>this work*</i>	928	769	75	874	874	874	2416	6810
	<i>Difference</i>	0	0	+25	+298	+298	+298	-3200	-2281
s-FF+cg	<i>based on [14]*</i>	928	511	50	389	389	389	351	2235
	<i>this work*</i>	928	511	75	687	687	687	2416	5991
	<i>Difference</i>	0	0	+25	+298	+298	+298	-200	-2281

in [14]. However, there the authors use a combination of clock-gating and scan-flip-flops (s-FF+cg), which results in storing costs of 6 GE per bit (plus a negligible overhead for clock gating logic). For ASIC prototyping it is sometimes not desirable to use clock gating, thus we decided to use D-flip-flops with enable signal (D-FF+en), which results in storage costs of 9 GE per bit.

In order to have a fairer comparison between our results and [14], we also report post-synthesis figures for a modified variant of their source code where we replaced the clock gating and scan-flip-flops with D-flip-flops with enable (9 GE) (see Table 1) We have also estimated the area requirements of our implementation using 6 GE scan-flip-flops in combination with clock gating.

Please note that the area of 387 GE for the *S-box* module in [14] comprises of both the shared S-box (359 GE) for the data path and the unshared S-box (28 GE) for the *Key schedule*. Thanks to a more optimized representation [4], the unshared PRESENT S-box we used only takes 22 GE, and since the unshared S-box is only used in the *Key Schedule* module we account its area share there. Though our post-synthesis results are smaller for the S-box, FSM, and top level glue logic (etc.), we have taken the figures from [14] to ensure a fair comparison.

As one can see, for the serialized architecture, the top level glue logic and the Key module are identical, while the control logic (FSM) is slightly more complex for our approach. Compared to [14] our approach requires six additional 4-bit wide 2-to-1 MUXes, which increase the area requirements of the storage components by 21 GE each. In total it is possible to save 130 GE, or around 37-40% of the area of the **S-box** by using our new approach. The overall savings are determined by the area share of the **S-box** module, which is dependent on the implementation strategy. For the serialized implementation described above, due to the minor contribution of the **S-box** module to the overall area requirements of only around 11-16%, in the end we achieve savings of only around 4-6%, while our estimates for a round-based implementation, where the area share of the **S-box** module is 62-68%, indicate overall savings of 25-27%. Table 2 summarizes these facts, and highlights that our new approach is more rewarding for architectures with a larger area share of the **S-box** module.

Table 2. Area savings for different implementation strategies.

Architecture Strategy	S-box Savings	Storage Strategy	S-box Area share	Overall Savings
serial	-37.0%	D-FF + en	11.4%	-4.2%
		s-FF + cg	15.7%	-5.8%
round-based	-40.6%	D-FF + en	61.8%	-25.1%
		s-FF + cg	67.9%	-27.6%

4 Experimental results

We analyzed power consumption traces obtained from a SASEBO G-II in order to evaluate the security of our new design. First, the measurement setup is introduced and a classical DPA analysis is performed to compare the results with [14] and prove that our implementation offers the same security level. Second, we demonstrate that the implementation of [14] is vulnerable against Mutual Information Analysis (MIA) in practice (as theoretically shown in [12]) and compare the strength of this attack with the correlation-enhanced collision attack targeting higher order moments as described in [10]. Last, we shortly revisit Wagner’s attack in the context of three-share Threshold Implementations.

4.1 Measurement setup

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 3MHz clock (from an on-board clock oscillator). The power consumption is measured over a 1Ω 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 1GS/s.

4.2 Side-channel resistance

We will now show that our implementation, which was presented in the previous section, has the same security level against the classical attacks as the implementation described in [14].

Fig. 3. Exemplary power trace of serialized implementation.

Figure 3 shows an exemplary power trace of the first round of an encryption run as well as a zoomed extract. The high peaks in the power consumption on the left side of Figure 3(a) are caused by the loading procedure of the plaintext and key to the cryptographic FPGA. The encryption starts at sample 8500 - for our analyses we omit these first 8500 samples. In Figure 3(b) one can clearly identify the peaks in the power consumption for every single clock cycle (333 samples between the peaks equal 3 MHz).

We conducted our evaluation based on 5,000,000 power traces. As in [14] we considered three different attack models for the DPA attack: *Hamming Weight* (HW) of one nibble of the S-box input, HW of one nibble of the S-box output and the *Hamming Distance* (HD) between two nibbles of consecutive S-box states. Since all attacks targeted 4 bits, 16 key guesses had to be analyzed. The results of the DPA attack for the three models are depicted in Figure 4. One can observe from the plots –as intuitively expected– that none of the attack models reveals the correct key hypothesis. Hence we can conclude that our implementation has no first order leakage.

In [14] the authors chose the HD of two consecutive state nibbles as the most promising leakage model. We found that attacking the HD between two

Fig. 4. DPA on sequential TI countermeasure implementation.

consecutive nibbles of the G -stage output (see Figure 1) shows higher leakage. Figure 5 shows exemplary correlation results (using known masks) comparing said models. When using the HD of two consecutive state nibbles as the model, one can nicely determine the repeating peaks caused by the shift to update the state, but the correlation coefficient is approximately five times lower than for the attack using our proposed model. We can only assume that this behavior is caused by a higher load of the output flip-flops of the G -stage, since the number of bits attacked and the number of bits which flip at the attacked point in time are the same for both. Hence, we chose the output of the G -stage as the target for our subsequent attacks. In addition, Figure 5 shows how many traces are needed until the correct key hypothesis is revealed. The original model needs approximately 2000 where as our proposed model needs only 500 traces.

As already mentioned we now want to investigate the resistance of the TI countermeasure against more sophisticated attacks. In [12] the authors showed, based on simulations, that it is possible to attack a Noekeon S-box secured by TI with MIA when all shares are processed in parallel. Unfortunately, a practical evaluation is still missing. Therefore, we can not say how efficient the attack is or if it works at all in practice, i.e., in the presence of noise. In [10] the author mounted a correlation-enhanced collision attack on the parallel implementation described in [14] and demonstrated that it can be practically broken targeting

Fig. 5. DPA results analysing 200,000 traces comparing two models, over number of traces (with known masks).

higher-order moments. We measured 10,000,000 traces of this implementation and mounted both attacks targeting the HD of consecutive outputs of the G -stage as described above (see Figure 6).

Fig. 6. Advanced attacks on parallel implementation of the TI countermeasure.

To the best of our knowledge this is the first successful practical attack of MIA targeting the TI countermeasure (see Figure 6(a)). Roughly 9,000,000 measurements are required until the correct key nibble is revealed. Figure 6(b) shows the result of a correlation-enhanced collision attack targeting second-order moments. With this attack the correct key nibble is already revealed after approximately 6,000,000 measurements which proves that this attack is more powerful than MIA in this scenario. We repeated both attacks against our implementation. Figure 7 depicts the results of both attacks, none of them succeeded. This is not a surprising result as our proposed model does not fit the implementation anymore. It should be noted though that while our S-box is serialized, there are still sources for univariate leakage in our implementation, e.g. the state update, which might be exploited with more appropriate models. To secure the implementation against these kind of attacks the whole implementation must be carefully serialized for every clock cycle, which is ongoing work.

Last, we want to revisit Wagner’s zero-offset attack described in [15]. The paper shows that it is theoretically possible to attack implementations where masks and masked values are processed in parallel, as it is the case for the parallel implementation of [14] which was attacked above. We performed the zero-offset attack against said implementation without success. Subsequently, we performed simulations of the zero-offset attack against TI with three shares and a classical masking scheme with two shares. As it turned out, the attack works against the two-share masking scheme (albeit it is very sensitive to noise) but it does not work against masking schemes utilizing three shares like TI. To be able to attack three shares one has to modify the attack, i.e., by raising the mean-free measurement values to the power of three instead of squaring.

Fig. 7. Advanced attacks on serialized implementation of the TI countermeasure.

However, for the attack to work we had to analyze 100 times more simulated power traces than for a MIA, which shows how sensitive this attack is against noise and why it does not work in practice.

5 Conclusion

In this paper we introduced two methodologies to efficiently implement 3-share TI to a given S-box. Applying these methodologies to the PRESENT S-box we were able to reduce its area requirement by 37-40% (130 GE), resulting in the smallest implementation of a protected PRESENT known in literature so far (2105 GE). Due to the small area share of the S-box in a serialized implementation, the overall savings were only around 4-6% at a timing overhead of a factor of 5.2. However, different implementation strategies, like round-based architectures, promise greater savings of more than 25% while having a smaller timing overhead of only 3.

We have practically shown that our approach achieves a similar security level as previously introduced implementations. In addition, we presented the first practical MIA against a parallel implementation of the TI and demonstrated that it is inferior to a correlation-enhanced collision attack using second-order moments.

It is also noteworthy to point out that our contribution allows to reduce the memory requirements of software implementations of S-boxes protected by the TI countermeasure by a factor of six. Future work will include investigations in this direction.

References

1. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In E. Prouff and P. Schaumont, edi-

- tors, *CHES2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76-91. Springer-Verlag, 2012.
2. P. Fiser and J. Hlavicka. Two-Level Boolean Minimizer BOOM-II. In *IWSBP2004*, pages 221-228, 2004.
 3. P. Fiser and J. Hlavicka. BOOM - A Heuristic Boolean Minimizer. *Computers and Informatics*, 22(1):19-51, 2003.
 4. J. Guo, T. Peyrin, A. Poschmann, and M.J.B. Robshaw. The LED Block Cipher. In B. Preneel and T. Takagi, editors, *CHES2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326-341. Springer-Verlag, 2011.
 5. Virtual Silicon Inc. 0.18 μm VIP Standard Cell Library Tape Out Ready, Part Number: UMCL18G212T3, Process: UMC Logic 0.18 μm Generic II Technology: 0.18 μm .
 6. P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Kobitz, editor, *CRYPTO1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104-113. Springer-Verlag, 1996.
 7. P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Weiner, editor, *CRYPTO1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388-397. Springer-Verlag, 1999.
 8. G. Leander and A. Poschmann. On the Classification of 4 Bit S-Boxes. In C. Carlet and B. Sunar, editors, *WAIFI2007*, volume 4547 of *Lecture Notes in Computer Science*, pages 159-167. Springer-Verlag, 2007.
 9. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, 2007.
 10. A. Moradi. Statistical Tools Flavor Side-Channel Collision Attacks. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 428-445. Springer-Verlag, 2012.
 11. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In P. Ning, S. Quing, and N. Li, editors, *ICICS2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529-545. Springer-Verlag, 2006.
 12. S. Nikova, V. Rijmen, and M. Schlaffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292-321, 2011.
 13. T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In J.R. Rao and B. Sunar, editors, *CHES2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172-186. Springer-Verlag, 2005.
 14. A. Poschmann, A. Moradi, K. Khoo, C. Lim, C. Wee, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *Journal of Cryptology*, 24(2):322-345, 2011.
 15. J. Waddle and D. Wagner. Towards Efficient Second-Order Power Analysis. In M. Joye and J.J. Quisquater, editors, *CHES2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 1-15. Springer-Verlag, 2004.