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

Sebastian Kutzner[1], Phuong Ha Nguyen[2], Axel Poschmann[2] and Huaxiong Wang[2].

[1] Laboratory of Physical Analysis & Cryptographic Engineering (PACE)
Temasek Laboratories @ NTU.
[2] Division of Mathematical Sciences,
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore

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 resolve many of the remaining open issues towards it's applicability. In particular, our contribution is three-fold: first we define which optimal (from a cryptographic point of view) S-boxes can be implemented with a 3-share TI. Second, we introduce two methodologies to efficiently implement these S-boxes. Third, as an example, we successfully apply these methodologies to PRESENT and are able to decrease the area requirements of its protected S-box by 57%.

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 many measurements. 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 increase the SNR ratio [10], to balance the leakage of different values [9] or to break the link between the processed data and the secret, i.e., masking [10]. Due to the presence of glitches masked implementation might still be vulnerable to DPA [10]. 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 [13].

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

more shares are required and the larger is the implementation. 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 Threshold Implementation to a larger degree function, this function must be represented as a composition of quadratic functions [13].

In this work we focus on 3-share Threshold Implementations of optimal 4-bit S-boxes. These S-boxes were defined in [8] and fulfill certain cryptographic properties which make them secure against cryptanalytic attacks. First we answer the question which of these optimal S-boxes can be protected using only 3-shares [1]. Second, we introduce two methodologies to efficiently implement these S-boxes in a 3-share TI scenario. We successfully apply these methodologies to the PRESENT S-box resulting in the smallest protected implementation known so far. Last, we investigate the security of our new design by practical measurements. During those investigations we develop a new attack model and, to the best of our knowledge, for the first time use the sum of square t-differences (SOST) [4] as a new distinguisher.

The remainder of this work is organized as follows. First we proof an open conjecture and recall important definitions in Section 2. Subsequently in Section 3, we introduce two new methodologies that allow to significantly reduce the area requirements of all TI S-boxes using the PRESENT S-box as an example. Section 4 describes the optimized hardware implementation of TI-PRESENT and its experimental analysis is performed in Section 5. Finally, Section 6 concludes the paper.

2 Decomposability of 4-bit S-boxes

The 3-share Threshold countermeasure can only be applied to permutations with a maximum degree of two [11,12]. Therefore, the decomposability of cubic 4-bit S-boxes into a composition of several quadratic vectorial boolean functions plays an important role when implementing the 3-share Threshold countermeasure. For example, in [13] the authors decompose the cubic PRESENT S-box into two quadratic vectorial boolean function $F(\cdot)$ and $G(\cdot)$ in order to apply the 3-share Threshold countermeasure.

In this section we prove the Nikova's conjecture presented at the poster session of CHES 2010. There the authors conjecture that any decomposable 4-bit S-box/permutation must belong to A_{16}, i.e., the alternating group of the 4-bit symmetric group S_{16}. A 4-bit S-box/permutation is considered as decomposable if and only if it can be written as a composition of several quadratic vectorial boolean functions. We recall some properties of a permutation in S_{16}.

Lemma 1. *A_{16} is a subgroup of S_{16}, i.e., if $p_1(\cdot)$ and $p_2(\cdot)$ are permutations in A_{16}, then the resulting permutation of their composition $p_3(\cdot) = p_1(p_2(\cdot))$ must be in A_{16} as well.*

[1] This result is independently found from that of http://eprint.iacr.org/2012/300.pdf which was accepted in CHES2012.

Lemma 2. *All linear and quadratic permutations in S_{16} are in A_{16}.*

Proof. In [13], the authors state that there are around 2^{26} quadratic permutations. Since the number of linear and quadratic permutations is not big, we can check the parity of all these permutations. If a permutation has a parity of $+1$, it belongs to A_{16}.

All parities of the considered permutations are $+1$. Hence, all these permutations belong to A_{16}.

Theorem 1. *If a permutation $p(\cdot)$ can be written as a composition of quadratic permutations, then $p(\cdot)$ is in A_{16}.*

Proof. The theorem is directly derived from the lemma 1 and lemma 2.

Corollary 1. *Theorem 1 implies that if a cubic permutation does not belong to A_{16}, it can not be written as a composition of several quadratic permutations.*

Note 1. The composition of a quadratic permutation and a linear permutation is again a quadratic permutation. Hence, a quadratic permutation is able to be decomposed in a composition of linear and quadratic permutations. This fact will be used for our improvement of the hardware implementation of the PRESENT S-box in the next sections.

Optimal and Decomposable 4-bit S-boxes An S-box is considered as optimal if it fulfills the following requirements [8]:

Definition 1. *Let $S : \mathbb{F}_2^4 \to F_2^4$ be an S-box. If S fulfills the following conditions we call S an optimal S-box:*

1. *S is a bijection,*
2. *$Lin(S) = 8$,*
3. *$Diff(S) = 4$,*

where $Lin(S)$ and $Diff(S)$ denote the linearity and differentiality of S-box S. The reader is referred to [8] for more detail on definitions of $Lin(S)$ and $Diff(S)$.

Definition 2. *Two S-boxes $S(x), S^{'}(x)$ are linearly equivalent iff there exist two 4×4-bit invertible matrices A, B and two 4-bit vector c, d such that*

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

Optimal S-boxes attract our attention due to their importance in designing cryptographic ciphers. An arbitrary optimal S-box always belongs to a certain class. In [8], the authors define 16 classes of linearly equivalent S-boxes in S_{16}. Each class can be represented by using one its S-box which is called representative of class.

Based on Note 1, if the representative of a considered class is decomposable, then all S-boxes in this class are decomposable as well, i.e., they belong to A_{16}. Checking the parity of the permutation of all class representatives reveals that exactly 8 classes (50%) are decomposable (see Table 1).

Note 2. The PRESENT S-box belongs to class 1. It implies that the PRESENT S-box is decomposable (which was shown in [13]).

Table 1. Decomposability of S-box classes.

Decomposable	0	1	2	4	5	7	8	13
Not decomposable	3	6	9	10	11	12	14	15

3 One S-box for all

In this section, we introduce a new methodology to improve the hardware implementation costs of the Threshold countermeasure. To illustrate our contribution we chose PRESENT as an example. Figure 1 shows how to apply the Threshold countermeasure to a 4-bit S-box: first it is decomposed into two stages G and F (horizontal), then each stage is shared (vertical). Figure 1 also shows that in [13] the authors implemented F and G using *six different* 8×4 vectorial Boolean functions $f_1, f_2, \ldots, g_3$. In the following we will show how to implement the same functionality with only *one* 8×4 vectorial Boolean function, this way significantly reducing the area/memory requirements of the S-box.

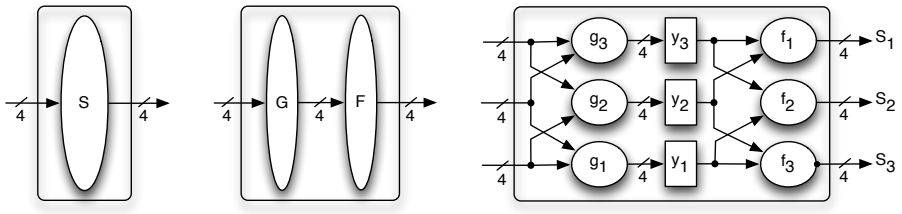


Fig. 1. Decomposition of an S-box [13]

3.1 The Horizontal Level

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

Lemma 3. *Assume a vectorial boolean function $S(\cdot) = G(G(\cdot))$, where $G(\cdot)$ is a vectorial boolean function. Then the hardware implementation of $S(\cdot)$ can be reduced by reusing the implementation of $G(\cdot)$.*

Proof. Experiments have shown that the costs for additional logic, e.g., a multiplexer, is less than implementing $G(x)$ twice. Numbers are provided in section 4.3.

The main problem of Lemma 3 is how to find a $G(x)$ such that $G(G(x))$ lies in the desired class, e.g., class 1 for the PRESENT S-box. We discovered that the only classes reachable by the construction $G(G(x))$ are 0, 1, 2 and 8. For class 1 we found the following quadratic $G(x)$ such that $S^{'}(\cdot) = G(G(\cdot))$.

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

The ANF of $G(x, y, z, w) = (g_3, g_2, g_1, g_0)$ is as follows:

$$\begin{aligned} g_3 &= x + yz + yw \\ g_2 &= w + xy \\ g_1 &= y \\ g_0 &= z + yw \end{aligned}$$

Using Definition 2 we know that the S-box of PRESENT $S(\cdot)$ is linearly equivalent to the found $S^{'}(\cdot) = G(G(\cdot))$, i.e

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

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

$$A = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,0 \\ 1\,0\,0\,0 \\ 1\,0\,1\,1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix}, \tag{1}$$

c and d are $(0001)_2 = 1$ and $(0101)_2 = 5$, respectively.

3.2 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 lemma:

Lemma 4. *The hardware templates of the vectorial boolean functions of $G(\cdot)$ are the same except for the indices of the inputs and the existence of constants.*

Proof. The lemma is derived from the construction of the vectorial boolean functions $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$. For example, if we take the latter constructed $G(x)$, then:

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

$$\begin{aligned}
g_{13} &= x_2 + y_2z_2 + y_2z_3 + y_3z_2 + y_2w_2 + y_2w_3 + y_3w_2 \\
g_{12} &= w_2 + x_2y_2 + x_2y_3 + x_3y_2 \\
g_{11} &= y_2 \\
g_{10} &= z_2 + y_2w_2 + y_2w_3 + y_3w_2
\end{aligned}$$

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

$$\begin{aligned}
g_{23} &= x_3 + y_3z_3 + y_1z_3 + y_3z_1 + y_3w_3 + y_1w_3 + y_3w_1 \\
g_{22} &= w_3 + x_3y_3 + x_1y_3 + x_3y_1 \\
g_{21} &= y_3 \\
g_{20} &= z_3 + y_3w_3 + y_1w_3 + y_3w_1
\end{aligned}$$

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

$$\begin{aligned}
g_{33} &= x_1 + y_1z_1 + y_1z_2 + y_2z_1 + y_1w_1 + y_1w_2 + y_2w_1 \\
g_{32} &= w_1 + x_1y_1 + x_1y_2 + x_2y_1 \\
g_{31} &= y_1 \\
g_{30} &= z_1 + y_1w_1 + y_1w_2 + y_2w_1
\end{aligned}$$

Therefore, we only need to implement $G_1(\cdot)$ and then reuse it for $G_2(\cdot)$ and $G_3(\cdot)$ by arranging the inputs appropriately.

Note that this technique can be applied not only for this special case but also in general whenever a function is shared. For example, let's take a look at the implementation presented in [13], stating the following ANFs for G_1, G_2 and G_3:

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

$$\begin{aligned}
g_{13} &= y_2 + z_2 + w_2 \\
g_{12} &= 1 + y_2 + z_2 \\
g_{11} &= 1 + x_2 + z_2 + y_2w_2 + y_2w_3 + y_3w_2 + z_2w_2 + z_2w_3 + z_3w_2 \\
g_{10} &= 1 + w_2 + x_2y_2 + x_2y_3 + x_3y_2 + x_2z_2 + x_2z_3 + x_3z_2 + y_2z_2 + y_2z_3 + y_3z_2
\end{aligned}$$

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

$$\begin{aligned}
g_{23} &= y_3 + z_3 + w_3 \\
g_{22} &= y_3 + z_3 \\
g_{21} &= x_3 + z_3 + y_3w_3 + y_1w_3 + y_3w_1 + z_3w_3 + z_1w_3 + z_3w_1 \\
g_{20} &= w_3 + x_3y_3 + x_1y_3 + x_3y_1 + x_3z_3 + x_1z_3 + x_3z_1 + y_3z_3 + y_1z_3 + y_3z_1
\end{aligned}$$

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

$$\begin{aligned}
g_{33} &= y_1 + z_1 + w_1 \\
g_{32} &= y_1 + z_1 \\
g_{31} &= x_1 + z_1 + y_1w_1 + y_1w_2 + y_2w_1 + z_1w_1 + z_1w_2 + z_2w_1 \\
g_{30} &= w_1 + x_1y_1 + x_1y_2 + x_2y_1 + x_1z_1 + x_1z_2 + x_2z_1 + y_1z_1 + y_1z_2 + y_2z_1
\end{aligned}$$

As one can see, our trick can also be applied to this implementation by handling the constants separately as $g_{i0}, g_{i1}, g_{i2}, g_{i3}$ comprise of similar monomials with different indices. Alternatively, it is possible to use correction terms, i.e., add the constant 1 to g_{22}, g_{21}, g_{20} and g_{32}, g_{31}, g_{30} such that the template of the terms match again.

4 Hardware Implementation

In the last sections we have described how to optimize the decomposition of cubic 4×4 S-boxes using the PRESENT S-box as an example. In this section we will describe an exemplary hardware implementation of PRESENT protected with the TI countermeasure with a shared data path and an unshared Keyschedule that is similar to profile 2 described in [13]. First we introduce the design flow used before we detail the hardware architectures and finally summarize the implementation results.

4.1 Design flow

For the hardware implementation in *VHDL*, we used the Boolean minimization tool `BOOM II` [1,2] 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 [14]. We used *Synopsys Power Compiler* version *E-2010.12-SP2* to estimate the power consumption of our ASIC implementations. For synthesis and for power estimation we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz. Note that the wire-load model used, though it is the smallest available for this library, still simulates the typical wire-load of a circuit with a size of around $10,000$ GE. We provide the power figures for information only and would like to highlight that it is not possible to compare them across different technologies.

4.2 Architecture and Design

Figure 2 depicts our architecture. The main differences between our design and profile 2 in [13] 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`, $\mathtt{m}_{d1}$ and $\mathtt{m}_{d2}$. 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 keyschedule.

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 staged instead of *six* as used in [13]. Recall, that we implement the PRESENT S-box $S(x)$ as $S(x) = A(G(G(Bx \oplus c)) \oplus d)$. Therefore, the inputs to G 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) until all three shares have been computed. Since, for the second stage, we do not need to process the input to G by `Bx+c`, we transform all three shares by $\texttt{B}^{-1}\texttt{(x+c)}$ (21 GE)[2] and store them in `I`, `II` and `III`. After the second stage is completed, the three shares are transformed by `Ax+d` (18 GE) and stored in the shift registers `State`, $\texttt{m}_{d1}$ and $\texttt{m}_{d2}$, which are shifting horizontally, and the new 4-bit nibbles are ready to be processed.

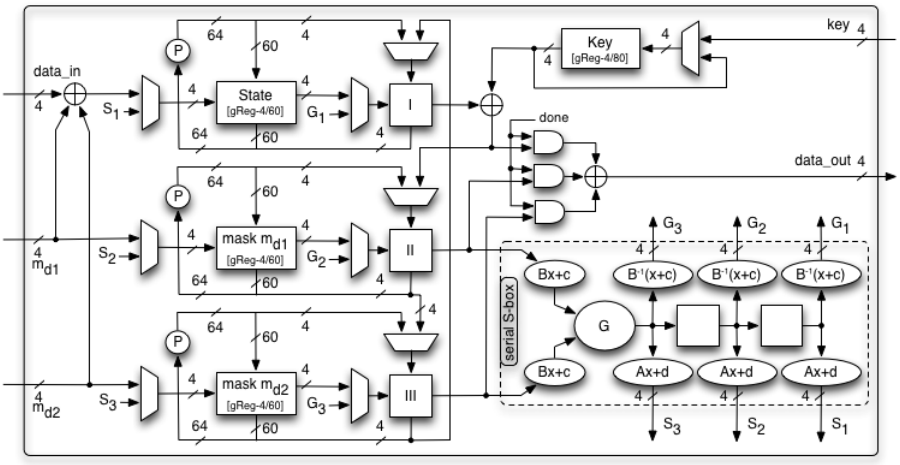


Fig. 2. Architecture of a serialized TI-PRESENT-80 using our new optimization techniques.

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

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

Table 2. Breakdown comparison of the post-synthesis implementation results of a serialized PRESENT-80 are shown in the upper half using D-flip-flops with enable (D-FF + en). The lower half shows estimated figures using scan-flip-flops and clock gating (s-FF + cg). All figures are Gate Equivalents (GE)

	Ref.	Etc.	Key	FSM	State	m_{d1}	m_{d2}	S-box	Sum
D-FF + en	[13]	58	778	139	587	587	587	351	3087
	this work	58	778	146	608	608	608	151	2957
	Difference	0	0	+7	+21	+21	+21	-200	-130
s-FF + cg (*estimated*)	[13]	58	520	139	389	389	389	351	2235
	this work	58	520	146	410	410	410	151	2105
	Difference	0	0	+7	+21	+21	+21	-200	-130

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 [13].

4.3 Performance figures

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 in [13]. However, there the authors use a combination of clock-gating and scan-flip-flops, 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, which results in storage costs of 9 GE per bit.

In order to have a fairer comparison between our results and [13], 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). The upper half of Table 2 shows these post-synthesis results. We have also estimated the area requirements of our implementation using 6 GE scan-flip-flops in combination with clock gating. This is shown in the lower half of Table 2.

Please note that the area of 387 GE for the *S-box* module in [13] comprises of both the shared S-box (359 GE) for the data path and the unshared S-box (28 GE) for the keyschedule. Thanks to a more optimized ANF [5] the unshared PRESENT S-box we used only takes 22 GE, and since the unshared S-box is only used in the *KeySchedule* module we account it's area share there. We have also taken into account that our post-synthesis results of the `S-box`, `FSM` and the top level glue logic (etc.) are smaller than the ones reported in [13] and estimated the figures accordingly.

As one can see, the top level glue logic and the `Key` module are identical in both architectures, while the control logic (`FSM`) is slightly more complex for our approach. Compared to [13] our architecture requires six additional 4-bit wide 2-to-1 MUXes, which increase the area requirements of the storage components

by 21 GE each. The `S-box` module is 57% smaller than the one reported in [13] yielding area savings of 200 GE. using our new approach in total it is possible to save 130 GE.

5 Experimental Results

In order to evaluate the security of our new approach, we analyzed power consumption traces obtained from SASEBO G-II. In the next subsection the measurement setup is introduced and subsequently the results of different DPA experiments are shown and compared to the results of [13]. In addition, we utilize additional techniques to investigate possible first order leakage. In the end we revisit the idea of Wagner et al. [15] which describes an attack targeting countermeasures where the masks and the masked state are processed simultaneously as it is usually the case for Threshold implementations.

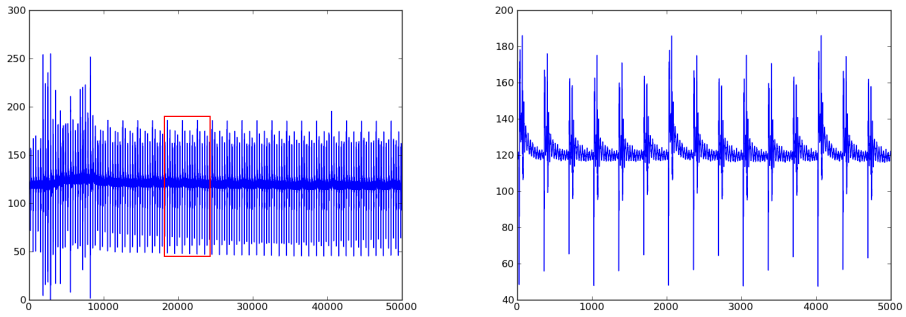
5.1 Measurement Setup

The SASEBO G-II hosts two FPGAs, i.e., one control FPGA (Xilinx XC3S400A-4FTG256, Spartan-3A series) and one cryptographic FPGA (Xilinx XC5VLX50-1FFG324, Virtex-5 series) which is decoupled from the rest of the board to minimize electronic noise from surrounding components. It is supplied with a voltage of 1V by an external stabilized power supply as well as with a 3MHz clock (24 MHz on-board clock oscillator utilizing a clock divider of 8). The power consumption is measured over a 1 Ω resistor inserted in the VDD line by using a differential probe. All power traces are collected with a LeCroy WR610Zi-s-32 at a sampling rate of 1GS/s.

5.2 Side-channel Resistance

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 at the left of Figure 3(a) are caused by the loading 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 (300 samples between the peaks equals 3 MHz).

To verify our measurement setup we first used 200,000 measurements and attacked our implementation knowing the random masks, i.e., we can guess intermediate masked values. Plaintexts and masks were chosen at random and are uniformly distributed. In [13] the authors chose the Hamming distance of two subsequent state nibbles as the leakage model. We think this model is not optimal since all 3*64 bit of the three states (`State`, $\mathtt{m}_{d1}$, $\mathtt{m}_{d2}$) are updated simultaneously. Hence, when attacking only one nibble, there is a lot of noise decreasing the correlation. We found that attacking the Hamming distance between two subsequent outputs of an S-box stage is more promising since here only 12 bit



(a) First round (b) Zoomed extract

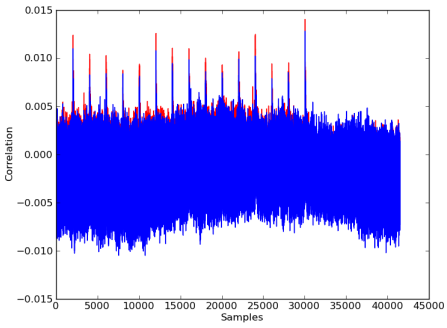
Fig. 3. Exemplary power trace

(3 shares * 4-bit S-box output) are updated simultaneously. Figure 4 shows the correlation results using the old model introduced in [13] and our new model. Using the old model one can nicely determine the 15 peaks representing the 15 updates of the state, i.e., the 15 shift operations, but the correlation coefficient is approximately five times lower than the one attacking the intermediate values between two S-box stages. The correct key guess becomes distinguishable after approximately 4,000 measurements.

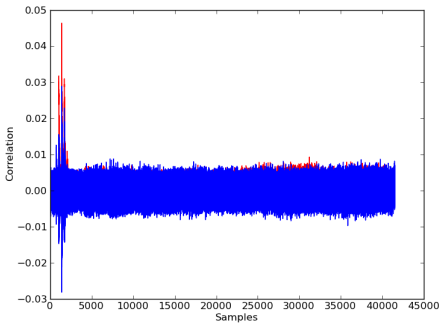
Next, we measured 5,000,000 traces. As in [13] we considered three different attack models for the DPA attack: HW of the S-box input, HW of the S-box output and the HD between two subsequent states. In addition we also considered our new model attacking the intermediate value between S-box stages. All attacks were performed nibble-wise, i.e., 16 key guesses had to be analyzed. Figure 5 shows the results of the DPA attack for the four models. As can be seen - and as expected - none of the attack models reveals the correct key nibble.

As mentioned in the introduction we want to extend our DPA analysis by utilizing additional measures to detect first-order leakage. We try to utilize the sum of square t-differences (SOST) introduced in [4]. Originally it was used to find points which contain the most information according to the chosen model in a template attack profiling phase. Here, we use it to see if there are any points containing any information (with a known key). The main advantage of SOST is, comparable to MIA [3], that it does not require a linear dependency between the attack model and the power consumption contrary to, e.g., the Pearson correlation coefficient.

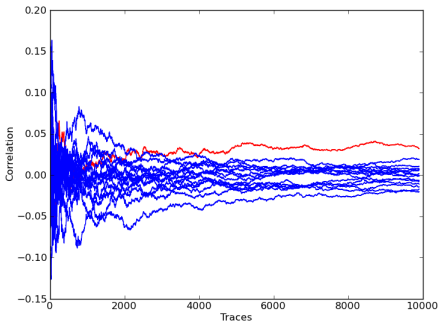
Subsequently, we tried SOST as a new DPA distinguisher. As classification function we chose the HD of two subsequent state nibbles. As one can see in Figure 6(a) the overall information content is very low. For comparison, Figure 6(b) shows the SOST trace, i.e., the information content targeting a plaintext nibble (note that for this analysis we included the first 8500 samples). Nonetheless, we performed a DPA attack using SOST as a distinguisher. Figure 6(c) shows



(a) HD of subsequent state nibbles

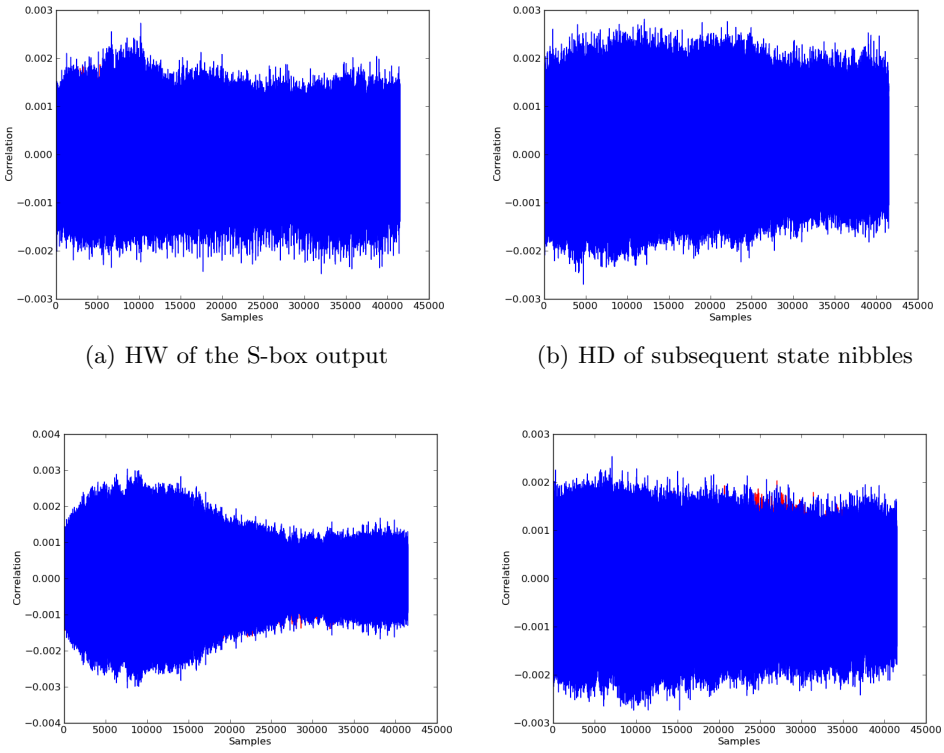


(b) HD of intermediate S-box outputs



(c) Number of traces at sample 1699

Fig. 4. DPA result with known masks



(a) HW of the S-box output

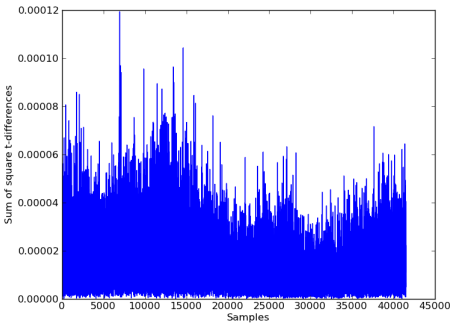
(b) HD of subsequent state nibbles

(c) HW of S-box input

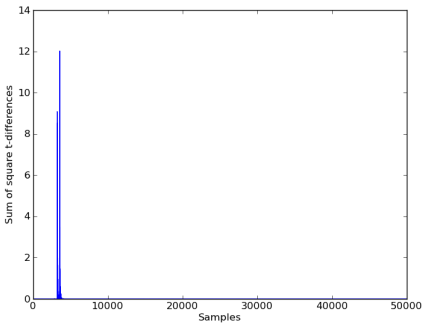
(d) HD of intermediate S-box outputs

Fig. 5. DPA results

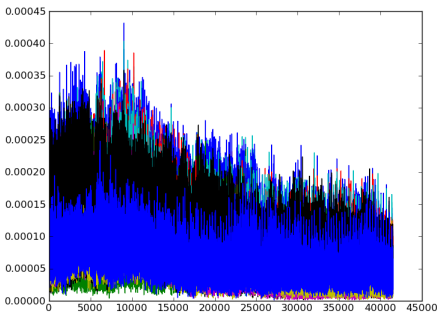
the results but as can be seen, there are no clear peaks indicating the correct key guess. To show that the idea indeed works and to highlight the strength of SOST as distinguisher we attacked the intermediate state with known masks using 200,000 measurements as in Figure 4. Figure 6(d) shows the result of this attack and as can be seen, the correct key hypothesis can be clearly identified and the relative difference between the highest and the second highest peak is much bigger than using the Pearson correlation coefficient. Hence, we think that in future work it is worth to evaluate the strength of SOST in more detail.



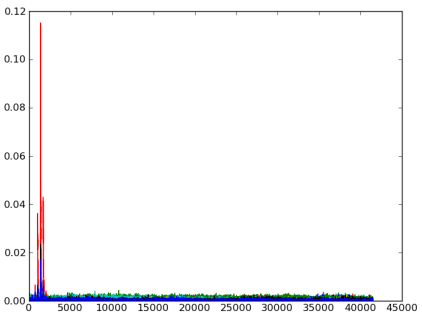
(a) HD of subsequent state nibbles using correct key hypothesis



(b) HW of plaintext



(c) DPA attack using SOST as distinguisher, correct hypothesis in red

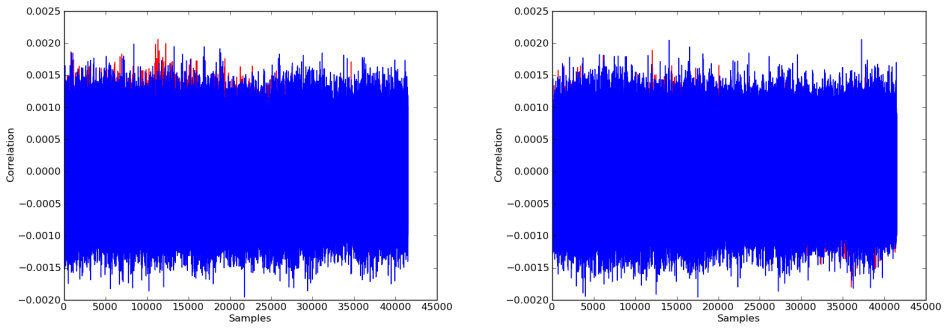


(d) DPA attack using SOST as distinguisher, correct hypothesis in red, known masks

Fig. 6. Results using the sum of square t-differences

Last, in 2004 Wagner et al. [15] introduced the Zero-offset attack for the, as they said, unlikely case that masked plaintexts and masks are processed at the same time. For the implementations in [13], our implementation, and especially Threshold Implementations in general, this case is true and hence these

implementations should be susceptible to this attack. Therefore, we took the previously measured 5,000,000 traces and performed the Zero-offset attack. Figure 7 shows the results of this attack using the before mentioned Hamming distance model. As we can see in 7 there are some red correlation peaks representing the correct key hypothesis rise above the rest. But repeating the attack for the second and third key nibble showed that the correct hypothesis cannot be distinguished. We repeated the attack using different models, i.e., targeting the intermediate state and using the Hamming weight, but none of the attacks worked. Simulations finally showed that the Zero-offset attack, i.e., squaring the power consumption, does not work with Threshold implementations. Future work will be done in this direction to find more suitable preprocessing functions.



(a) HD of subsequent state nibbles, key byte 1 (b) HD of subsequent state nibbles, key byte 2

Fig. 7. DPA results of the Zero-offset attack

6 Conclusion

In this paper we have proven that all optimal S-boxes which can be protected by the 3-share Threshold countermeasure belong to A_{16}. Furthermore, we introduced two methodologies to efficiently implement these S-boxes in a TI scenario. Applying these methodologies to the PRESENT S-box we were able to reduce its area requirement by 57% (130 GE), resulting in the smallest implementation of a protected PRESENT so far (2105 GE). Finally, we have proven the security of our new design by practical experiments.

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

References

1. P. Fišer and J. Hlavička. BOOM - A Heuristic Boolean Minimizer. *Computers and Informatics*, 22(1):19–51, 2003.
2. P. Fišer and J. Hlavička. Two-Level Boolean Minimizer BOOM-II. In *Proceedings of 6th Int. Workshop on Boolean Problems – IWSBP'04*, pages 221–228, 2004.
3. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
4. Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In *Proceedings of the 8th international conference on Cryptographic Hardware and Embedded Systems*, CHES'06, pages 15–29, Berlin, Heidelberg, 2006. Springer-Verlag.
5. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The led block cipher. In *CHES*, pages 326–341, 2011.
6. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. pages 104–113. Springer-Verlag, 1996.
7. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.
8. G. Leander and A. Poschmann. On the classification of 4 bit s-boxes. In *Proceedings of the 1st international workshop on Arithmetic of Finite Fields*, WAIFI '07, pages 159–176, Berlin, Heidelberg, 2007. Springer-Verlag.
9. Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In *Systems CHES 2005, 7th International Workshop*, pages 172–186. Springer, 2005.
10. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
11. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
12. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *Journal of Cryptology*, October 2010.
13. Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2,300 ge. *J. Cryptol.*, 24(2):322–345, April 2011.
14. 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: 0.18μm, July 2004.
15. Jason Waddle and David Wagner. Towards efficient second-order power analysis. In *CHES*, pages 1–15, 2004.