Higher-Order Threshold Implementation of the AES S-box

Thomas De Cnudde[1], Begül Bilgin[1], Oscar Reparaz[1], Ventzislav Nikov[2], and Svetla Nikova[1]

[1] KU Leuven, ESAT-COSIC and iMinds, Belgium
{name.surname}@esat.kuleuven.be
[2] NXP Semiconductors, Belgium venci.nikov@gmail.com

Abstract In this paper we present a threshold implementation of the Advanced Encryption Standard's S-box which is secure against first- and second-order power analysis attacks. This security guarantee holds even in the presence of glitches, and includes resistance against bivariate attacks. The design requires an area of 7849 Gate Equivalents and 126 bits of randomness per S-box execution. The implementation is tested on an FPGA platform and its security claim is supported by practical leakage detection tests.

Keywords: Higher-Order, Threshold Implementations, AES, S-box, Masking

1 Introduction

Side-Channel Analysis (SCA) and more specifically Differential Power Analysis (DPA) [12] are considered to be powerful methods which can be used to extract secrets, e.g. keys or passwords, from cryptographic implementations running on embedded devices. The wide usage of these devices demands strong yet practical methods to mitigate this problem. A sound and popular such method is masking [6, 11]. Masking works by splitting every intermediate variable that depends on the secret into several shares such that knowledge of any share does not provide any information about the intermediate variable. This splitting breaks the dependency between the average instantaneous power consumption and the sensitive intermediates handled by the implementation, and thus thwarts first-order DPA attacks.

In theory, however, a masked implementation can always be broken by a higher-order attack. Higher-order attacks consider information from several shares simultaneously and are increasingly difficult to mount as the order increases, both in terms of number of traces [6] and computational complexity. Nonetheless, second-order attacks have been shown to be practical to mount [14, 19, 20, 24, 27, 28] and hence its protection is of importance. Higher-order masking schemes provide security guarantees against higher-order DPA attacks under specific assumptions, and up to a certain order.

When implemented in hardware, masking can lead to insecure designs due to glitches. Standard CMOS gates can glitch, and these glitches can cause the power consumption to depend on unmasked variables. This behavior degrades the security claims. For instance, Mangard et al. [13] present first-order attacks against masked implementations in hardware exploiting this idea.

Threshold Implementation (TI) [17, 18] is a specific masking approach that provides security even in the presence of glitches in the hardware. First-order TIs of the Advanced Encryption Standard (AES) have been shown to being practically feasible as well as being secure [2,4,16]. The theory of TI has recently been extended to provide higher-order security by Bilgin et al. [3].

Prouff and Roche's Higher-Order Glitches Free Implementation (HOGFI) [21] provides an alternative approach to TI. A first-order secure HOGFI of the AES S-box has been presented [15]. However, a higher-order extension has not yet been put into practice for AES. To our knowledge, the only higher-order implementation of this method is applied to PRESENT [9].

Contribution. We provide the first higher-order threshold implementation of the AES S-box. Our design shows up to second-order security (including bivariate attacks) in the presence of glitches. This paper is, to our knowledge, the first one to show this security in practice within the context of TI. Additionally, we discuss several trade-offs between randomness and area that can be considered.

Organization. Section 2 introduces our notation, the necessary background information regarding higher-order TI and Canright's decomposition of the AES S-box on which we base our implementation. In Section 3, we present our hardware design of which the implementation costs are given in Section 4. Discussions of these results by comparing them with other glitch resistant implementations of the AES S-box and by investigating trade-offs in area and randomness through different design decisions are also given in the same section. We detail our measurement setup and the results of the side-channel analysis in Section 5. Finally, the conclusion is drawn in Section 6.

2 Preliminaries

In this section, we first introduce our notation, then provide a brief description of the threshold implementation technique to produce higher-order masked hardware implementations, and finally we end with the description of a compact (unmasked) implementation of the AES S-box that will serve as a basis of our masked implementation.

2.1 Notation

We use lower-case characters to denote elements in $\mathrm{GF}(2^n)$. A function f is defined from $\mathrm{GF}(2^n)$ to $\mathrm{GF}(2^m)$ and can be considered as an m-tuple of Boolean functions $(f^1(x), \ldots, f^m(x))$, where $x \in \mathrm{GF}(2^n)$. Similarly, $x \in \mathrm{GF}(2^n)$ can be denoted as $(x^1, \ldots, x^n)$, where $x_i \in \mathrm{GF}(2)$. We use $\oplus$ for XOR and $\otimes$ for multiplication in a given field. If the multiplication is bit-wise, we drop $\otimes$.

In order to perform a masked computation, a secret variable x should be split into s_x shares x_i. In this paper, we consider Boolean masking for this initial split which is described as follows: without loss of generality, the shares $x_1, ..., x_{s_x-1}$ are drawn from independent and uniform random distributions and the share x_{s_x} is calculated s.t. $x = \bigoplus_i x_i$ holds. A shared vector (sharing) (e.g. $(x_1, \ldots, x_{s_x})$) is denoted by bold characters (e.g. $\mathbf{x}$). A sharing is a uniform masking if for each value x, the corresponding vectors with masked values occur with the same probability.

In order to perform operations in the masked domain, the function f is also split in shares f_i which are called component functions. The sharing of f is denoted by $\mathbf{f}$.

2.2 Threshold Implementations

Threshold implementation (TI) is a masking method which provides security against higher-order DPA (hence the name higher-order TI). It diverges from many other masking schemes since it can provide security when non-ideal, glitchy cells are used given the following property:

$d^{th}-$order non-completeness. Any combination of up to d component functions f_i of $\mathbf{f}$ must be independent of at least one input share x_i.

This property enforces the combination of leakages from the calculation of d component functions to be independent of the sensitive variable x given a uniform sharing $\mathbf{x}$. We refer the reader to [3] for details. In addition, it has been shown that there always exists a d^{th}-order non-complete sharing of a degree t function f with $s_{in} \geq td + 1$ input shares [3]. This naturally implies that the required number of shares for a given security increases together with the degree of the function.

In [3], a method for generating the component functions with $s_{in} = td + 1$ input and $s_{out} = \binom{s_{in}}{t}$ is provided. Hereon, we refer $\mathbf{f}$ with s_{in} input and s_{out} output shares as (s_{in}, s_{out}) sharing.

Uniform sharings vs. refreshing. As stated in Section 2.2, the computation of a sharing $\mathbf{f}$ requires the input $\mathbf{x}$ to be uniform. However, the fact that $\mathbf{x}$ is uniform does not automatically ensure that $\mathbf{y}$ is uniform. The lack of uniformity of $\mathbf{y}$ poses a problem if this variable is plugged into another sharing $\mathbf{g}(\mathbf{y})$, since the input $\mathbf{y}$ should also be uniform for $\mathbf{g}$ to be secure. By careful selection of the shared function $\mathbf{f}$, it is possible to guarantee the uniformity of $\mathbf{y}$ given a uniform input $\mathbf{x}$. We refer to such a sharing $\mathbf{f}$ as a uniform sharing. This uniformity allows an elegant composition mechanism: uniform sharings can be composed freely[3] without using further randomization, and still provide first-order security for the

[3] We mean strict composition as $g \circ f$. If g sees the concatenation of two functions f_1, f_2, one should make sure that the input to g stays uniform. This does not automatically happen even if f_1 and f_2 are independently uniform [4].

whole circuit [18]. If one cannot find uniform sharings, it is always possible to resort to refreshing the sharing $\mathbf{y}$ prior to applying $\mathbf{g}$. This refreshing produces a uniform output at the cost of additional randomness.

The situation in higher-order threshold implementations is more subtle. It has been shown in [23] that the composition of uniform sharings without refreshing is not necessarily higher-order secure; the composition can be made higher-order secure by introducing a refreshing block. Thus, in our design, we refresh the output of each non-linear function to provide higher-order security.

2.3 Canright's Very Compact AES S-box

The AES S-box is an 8-bit permutation composed of a multiplicative inversion in GF(2^8) followed by a GF(2)-affine transformation [8]. Side-channel resistant implementations of this S-box are commonly based on subfield arithmetic as proposed by Rijmen [25] and explored by Canright [5]. This approach typically produces low-area circuits. It takes its name from the recursive decomposition of the S-box into computations in smaller fields. Namely, the GF(2^8) inversion is first decomposed into arithmetic operations in GF(2^4); and in turn the nonlinear operations are performed in the subfield GF(2^2). The resulting computation is composed of a GF(2)-linear (LM) and inverse GF(2)-linear map (ILM), several GF(2^2) multiplications, bitwise XORs and multiple instantiations of linear operations in GF(2^2) (l_i). For a detailed description of the individual operations, we refer to the original work [5].

3 Hardware Implementation

This section gives an overview of the choices we made during the design of the second-order threshold implementations of the AES S-box.

3.1 Redefining the S-box decomposition

Converting the Canright S-box to a threshold implementation can be achieved on several levels. Each individual block can be composed with the neighbouring blocks or decomposed into smaller sub-blocks to attain different trade-offs between area, speed and randomness. We acknowledge the fact that randomness requirements also (indirectly) affect the area requirements. Hence, we strive for a compact, low-area implementation, and at the same time we try to keep the randomness requirements as low as possible.

For the discussion of our shared AES S-box, we rely on Figure 1. We choose to implement the square scale and multiplication operations in GF(2^4) as done by Bilgin et al. [2]. This adaptation requires less randomness and clock cycles than sharing their subfield functions in GF(2^2) since some of the refreshing and registering that must follow the nonlinear operation is avoided. The inversion in GF(2^4) is of algebraic degree three. Since no small second-order non-complete sharing of such a function has been yet proposed, we share its subfield decomposition, which is contained of a linear operation and three multiplications in GF(2^2). Although, this increases the number of clock cycles by one, it keeps the area and randomness contained.

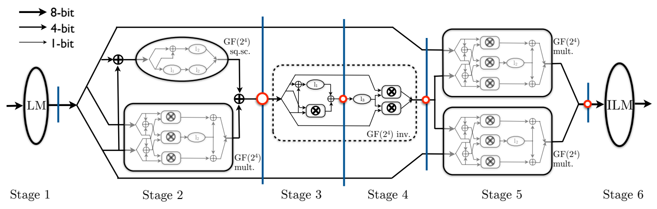

Figure 1. Unmasked S-box based on Canright

We decompose the calculation of the S-box into 6 pipeline stages. All stages are separated by registers, indicated by the vertical blue lines in Figure 1, in order to satisfy the non-completeness property within pipeline stages as explained in Section 2.2. Note that the register after the linear map and the register preceding the inverse linear map can be merged with the AES state or key registers.

3.2 Sharing the Nonlinear Operations

The GF(2^4) and GF(2^2) multipliers are the only nonlinear operations of our S-box. For the second-order threshold implementation we need a sharing with second-order non-completeness. There are two known sharings with $s_{in} \geq td+1$ input shares and fulfilling the non-completeness condition for a function of the form $f(x, y) = xy$, with $x, y \in$ GF(2) [1,3]. One uses $s_{in} = 5$ input shares and results in $s_{out} = 10$ output shares, the other one accepts $s_{in} = 6$ input shares and outputs $s_{out} = 7$ shares. We choose the $(6, 7)$-sharing for the multiplications with the following observations in mind:

- To achieve higher-order security, all nonlinear sharings need refreshing of all their output shares as noted in Section 2.2. Hence, we remask the outputs of all the multiplications. The details of how this refreshing is done is described in Section 3.4. The lower s_{out}, the lower the consumed randomness is. By using only 7 output shares, the required randomness can be decreased by 30% in comparison to using 10 output shares.
- The (5,10)-sharing requires 20 XOR gates, 25 AND gates and 10 output registers per bit. For one bit of the (6,7)-sharing, 29 XOR and 36 AND gates are needed while 7 output registers suffice. Taking 5.66 Gate Equivalents (GE) per register bit, 1.33 GE per AND gate and 2 GE per XOR gate, this boils down to 12% increase when using 6 instead of 5 input shares. These GE numbers are obtained from the NanGate 45nm Open Cell Library.

With our choice of the $(6,7)$-sharing over the $(5,10)$-sharing we pay a slightly larger area for a substantial reduction of the required randomness.

We use the following (6,7)-sharing for all the bit-wise multiplications $a = f(x,y) = xy$ that occur in the Boolean function description of the field multipliers.

$$\begin{aligned}
a_1 &= x_2y_2 \oplus x_1y_2 \oplus x_2y_1 \oplus x_1y_3 \oplus x_3y_1 \oplus x_3y_2 \oplus x_2y_3 \\
a_2 &= x_3y_3 \oplus x_3y_4 \oplus x_4y_3 \oplus x_3y_5 \oplus x_5y_3 \\
a_3 &= x_4y_4 \oplus x_2y_4 \oplus x_4y_2 \oplus x_2y_6 \oplus x_6y_2 \\
a_4 &= x_5y_5 \oplus x_1y_4 \oplus x_4y_1 \oplus x_1y_5 \oplus x_5y_1 \\
a_5 &= x_2y_5 \oplus x_5y_2 \oplus x_4y_5 \oplus x_5y_4 \\
a_6 &= x_6y_6 \oplus x_3y_6 \oplus x_6y_3 \oplus x_4y_6 \oplus x_6y_4 \\
a_7 &= x_1y_1 \oplus x_1y_6 \oplus x_6y_1 \oplus x_5y_6 \oplus x_6y_5
\end{aligned}$$

For convenience, we provide the Boolean functions descriptions of the field multipliers for which we denote each element with its most significant bit on the left-hand side. GF(2^4) multiplication $(a^1, a^2, a^3, a^4) = (x^1, x^2, x^3, x^4) \otimes (y^1, y^2, y^3, y^4)$:

$$\begin{aligned}
a^1 &= x^1y^1 \oplus x^3y^1 \oplus x^4y^1 \oplus x^2y^2 \oplus x^3y^2 \oplus x^1y^3 \oplus x^2y^3 \oplus x^3y^3 \oplus x^4y^3 \oplus x^1y^4 \oplus x^3y^4 \\
a^2 &= x^2y^1 \oplus x^3y^1 \oplus x^1y^2 \oplus x^2y^2 \oplus x^4y^2 \oplus x^1y^3 \oplus x^3y^3 \oplus x^2y^4 \oplus x^4y^4 \\
a^3 &= x^1y^1 \oplus x^2y^1 \oplus x^3y^1 \oplus x^4y^1 \oplus x^1y^2 \oplus x^3y^2 \oplus x^1y^3 \oplus x^2y^3 \oplus x^3y^3 \oplus x^1y^4 \oplus x^4y^4 \\
a^4 &= x^1y^1 \oplus x^3y^1 \oplus x^2y^2 \oplus x^4y^2 \oplus x^1y^3 \oplus x^4y^3 \oplus x^2y^4 \oplus x^3y^4 \oplus x^4y^4
\end{aligned}$$

GF(2^2) multiplication $(a^1, a^2) = (x^1, x^2) \otimes (y^1, y^2)$:

$$\begin{aligned}
a^1 &= (x^1 \oplus x^2)(y^1 \oplus y^2) \oplus x^1y^1 \\
a^2 &= (x^1 \oplus x^2)(y^1 \oplus y^2) \oplus x^2y^2
\end{aligned}$$

3.3 Sharing the Linear Operations

Linear operations are well-known to be easy to mask. The linear computation is performed on each share independently. This works for the following functions:

- Square scale in GF(2^4)
- l_1 and l_3 in GF(2^2)
- Linear map and inverse linear map

In our implementation, we chose to instantiate $s_{in} = 6$ copies for each of these functions.

The affine operations are performed in parallel with the multiplication in Stages 2 and 3. The output of the affine operation is added to the output of the multiplication; the novelty here is that we can add them *before* storing them in the register, and thus only store the result of the addition. In this way we use less registers and hence a lower area. This addition has to be performed carefully. We have to ensure that the output of the addition still satisfies non-completeness. For instance one wire can carry the value $a_1 \oplus A(x^2, y^2)$ instead of a_1, where A symbolizes an affine operation. This process can be alternatively seen as the sharing of a single function $f' = x \otimes y \oplus A(x, y)$.

3.4 Mask Refreshing and Compression

Apart from the initial sharing, extra randomness is required in the refreshing blocks to attain second-order security. We use a ring structure for this refreshing as proposed in [23]. An advantage of this method is that the sum of the fresh masks does not need to be saved in an extra register. Since we operate on six input shares in each stage, we need to *compress* the seven preceding output shares into six shares. Figure 2 shows how the mask refreshing and the compression are performed over register boundaries after each nonlinear operation. The points where these refreshing and compression layers occur are depicted in Figure 1 by red circles.

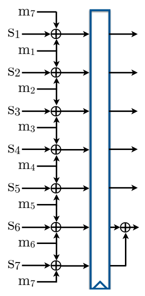

Figure 2. Ring refreshing and compression

4 Implementation Cost and Trade-Offs

In this section, we elaborate on the implementation cost of our design and the impact of possible trade-offs. We use Xilinx ISE version 12.2 to verify the functionality of our design and Synopsys 2010.03 with the NanGate 45nm Open Cell Library when providing area estimations.

4.1 Implementation Cost

Table 1 shows the summary of the implementation cost for our S-box design. This is compared to previous first-order secure TIs of the AES S-box. We now briefly discuss these figures.

Table 1. Implementation cost of different TIs of the AES S-box

S-box	Area [GE]*	Randomness [bit]	Clock Cyles	Security
[16]	-/4244	48	5	1^{st}-order
[2]	3708/3003	44	3	1^{st}-order
[4]	2835/2224	32	3	1^{st}-order
This Paper	11174/7849	126	6	2^{nd}-order

*: Using `compile` / `compile_ultra` synthesis option

Area Requirements. From Table 1 we can see that our design uses $\times 1.84$ more area when compared to the first-order TI from [16], or $\times 3.53$ more area when compared to the more compact first-order secure design of [4]. Both figures for the synthesis options "compile" and "compile ultra" are provided[4]. This is the price we pay in area to go from first-order security to second-order security.

Table 2 lists the contribution in area of the several components from a single S-box. Note that the l_3 operation, which is the inversion in GF(2^2), is merely a swapping of wires and has therefore no contribution to the area. Also listed are the numbers for the second-order TI of the full AES based on [16]. As in [4], we use $d+1$ shares in the key and state array. The registers that hold the randomness are not included in the figures. Note that the full AES was not tested in practice. In addition, Table 2 also lists the results of the synthesis for a Virtex 5 FPGA. We provide these figures for future comparison, following the HOGFI design from [15]. Currently, it is hard to discuss the impact of scaling from first-order to second-order security for HOGFI. An extrapolation factor of $\times 1.667$ could be used[5], but may be too optimistic, e.g. for the PRESENT S-box, the factor of area increase was shown to be $\times 2.3$ in [9]. Furthermore, the area of the second-order PRESENT S-box implementation amounts to 8338 GE, which is larger than our second-order design.

Randomness Requirements. Our design requires four ring refreshing on seven shares. Two on 4 bit shares, one on 2 bit shares and one on 8 bit shares. This results in 126 bits of randomness per S-box execution. For a full AES execution we require 3.814 kB, this includes the initial sharing and the randomness needed to increase the three shares of the state and key arrays to the six input shares of the S-box.

When compared to other TIs that provide first-order security, our implementation requires $\times 2.6$ more randomness than [16] or $\times 3.9$ more than [4].

For the first-order HOGFI implementation [15], 432 bits are consumed per S-box execution. This increases to 1440 bits for the second-order HOGFI of the

[4] Special attention is paid so that these options optimize within, but not across, block boundaries. Otherwise, the non-completeness property could be destroyed by the synthesis tool.

[5] This is because the size of a HOGFI circuit grows, very roughly, with the number of shares $2d+1$.

Table 2. Area of different functions of the masked S-box

	Area [GE] Compile	Area [GE] Compile Ultra
Linear Map	22.3	22.3
Inverse Linear Map	17.6	17.6
GF(2^4) Square Scale	11.2	11.2
GF(2^2) l_1	4.8	4.8
GF(2^2) l_3	0	0
GF(2^4) Multiplier	2189.0	1117.7
GF(2^2) Multiplier	408.8	248.6
Registers	1937.77	1937.77
Control S-box	799.6	797.2
Total S-box	11173.57	7849.27
AES Key & State Array	7204.8	6807.75
AES Control	223.2	215.2
Total AES	18601.57	14872.22
	Virtex 5 FPGA	
Total S-box [FFs]		691
Total S-box [LUTs]		3696
Total S-box [Slices]		1925

*: These numbers are for the full, untested AES

AES S-box [6]. Thus, our implementation consumes $\times 11.4$ less randomness than a second-order HOGFI S-box.

Clock Cycle Requirements. Our S-box currently evaluates in six clock cycles. This is one more than [16] and three clock cycles more than the fastest TI [4]. In Section 4.2, we show that our S-box can easily be modified to achieve an evaluation speed of only four clock cycles. For the S-box implemented with HOGFI, 132 clock cycles are required for the first-order implementation. This scales to 220 cycles for the second-order implementation. This is substantially larger than our design.

4.2 Directions for Optimizations

We now list several trade-offs we can make to reduce the area and their effect on the randomness.

– We can reduce the area by changing the number of instances of the linear functions to $d + 1 = 3$. This was done in the *nimble version* of [4]. The required randomness is not changed by this modification.

[6] This number is obtained by applying the HOGFI theory [21].

– The registers for the output of the linear map and the input of the inverse linear map can be bypassed. This was shown in [2, 4, 16]. This bypass will not lead to a reduction in the randomness cost. The execution speed will however be improved by two clock cycles, making a whole AES encryption with this S-box as fast as the implementation of [16]. Note that in a full AES implementation, these two Stages do not add an area overhead, as these registers can be merged with the State and Key Arrays.
– As previously mentioned, the (5,10)-sharing of [3] can be used to save area, but this will increase the required randomness to 180 bits.

5 Side-Channel Analysis Evaluation

For the purpose of evaluating the security claims of our approach, we implemented the design on a Virtex-II xc2vp7-fg456-5 FPGA on a SASEBO-G board. We generate the required randomness prior to the S-box execution on the control FPGA on board. We silence all activity on the board except the S-box lookup during the lookup itself. The design is clocked at 3.072 MHz and the instantaneous power consumption is acquired with a Tektronix DPO 7254C oscilloscope at 500 MS/s. The platform is very low noise.

Methodology. We pursue the following steps to test that the soundness of our setup and masking. First, we switch off the randomness source. This effectively disables the masking countermeasure, thus the design is expected to be vulnerable. Nevertheless, the analysis is first performed in this setting to show that the setup is sound. Then, we repeat the analysis with the randomness source switched on. Any gain in resistance shown in the analysis is then exclusively due to masking.

Security claims. We claim security against first- and second-order power analysis attacks. That is, the adversary is bounded in the statistical order that he can use in the attacks. Note that since higher statistical moments are increasingly more difficult to estimate in the presence of noise, higher-order attacks beyond our security claims requires considerably more traces to work on and thus are deemed more impractical.

There is an orthogonal classification dividing attacks into uni- or multi-variate ones according to the number of different time samples considered jointly in the analysis. This distinction is relevant in practice since multi-variate attacks can be more cumbersome to mount. Here, our second-order claims also include the bi-variate case as the variables considered can be evaluated in different times.

Leakage detection. We use leakage detection to test our security claims [7,10,26]. A basic leakage detection to test the univariate first-order security claim is as follows. Two sets of measurements are acquired corresponding to a lookup of either a fixed value or a random value. A statistical test is applied to test the null hypothesis "the means of the two trace distributions are the same". We use Student's t-statistic and compare it against a threshold of ± 5 corresponding to a confidence level $> 99.999\%$. If the statistic surpasses the threshold, there is a statistically significant difference in the means, and thus the means carry some information on the handled value. The test is failed in this case and passed otherwise.[7]

Higher statistical orders are tested in a similar fashion by first pre-processing the traces with an appropriate function. In our case, we use the centered product [6]. If no observable leakage is detected, classical key-recovery attacks will not succeed.

5.1 Univariate analyses

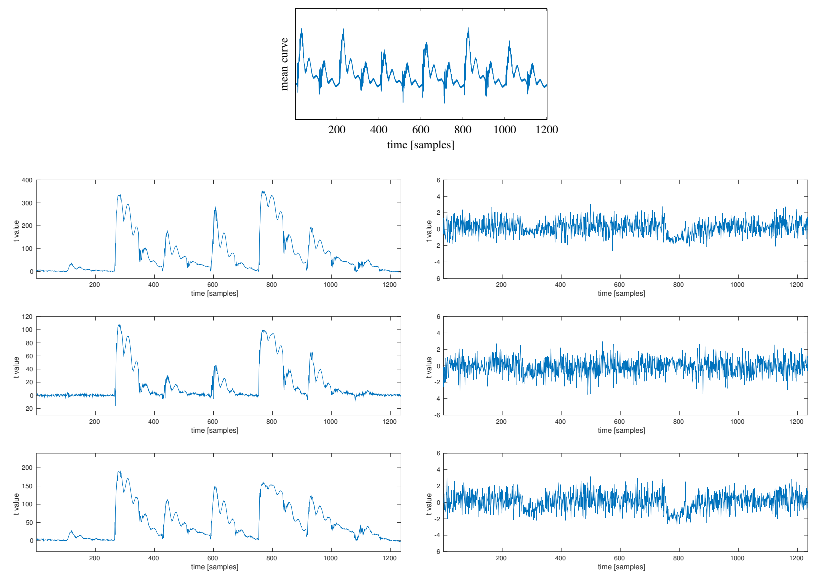

Figure 3. Univariate analysis. Top Figure: power trace of one S-box execution, Lower Figure: top to bottom, left to right. a) first-order analysis, masks off b) second-order analysis, masks off c) third-order analysis masks off d) first-order analysis, masks on e) second-order analysis, masks on f) third-order analysis, masks on. Note that although the S-box evaluates in 6 cycles, we raise the trigger 1 cycle before the actual computation.

[7] Usual precautions should be taken when mounting the test. For instance, in order to assure that no environmental factor creates an undesired balance between the sets, we interleave the lookups from each set in a random manner.

Masks off. In Figure 3, we plot the t-statistic when the masks are switched off. We used 500 000 traces to produce plots with clear leakage which was more than the required amount of traces for detecting leakage. We can see clear excursions of the statistic beyond the specified threshold of ± 5. The same behavior can be observed for the second and third moment. Hence, all three tests fail. This is expected and means that the measurement setup is sound.

Masks on. When the randomness is switched on, the implementation passes the leakage detection tests for first and second order using up to 100 million traces. This supports the claim of univariate first- and second-order univariate security of the implementation.

5.2 Bivariate analysis

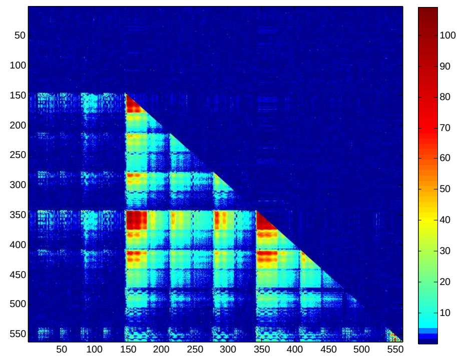

Figure 4. Bivariate analysis with a) masks off in lower left corner, and b) masks on in upper right corner. Best viewed in color. We plot the absolute value of the t-statistic.

The previous analysis can be straightforwardly generalized for multivariate statistics. Each time sample is centered before combined with all other timesamples. The combination function is centered product. To alleviate the computational effort, we sample at 200 MS/s to get $\times 2.5$ shorter traces.

Masks off. Figure 4 shows the result of the bivariate second-order test when the masks are off using 100 000 traces. Clear excursions above ± 5 are seen, both in the diagonal (squaring) and off the diagonal (centered product of two different time samples). This serves as confirmation that our approach is sound, in particular, the change in the sampling rate did not have a significant effect on the magnitude of the t-statistic.

Masks on. We repeated the same test when the randomness is switched on. The implementation shows resistance when using up to 70 million measurements. No excursion beyond the threshold were detected, and thus the test is passed.

6 Conclusion

In this work, we presented the first higher-order threshold implementation of AES S-box with second-order univariate and bivariate security in the presence of glitches. We directed our attention to the S-box specifically as it is the hardest part to secure due to its nonlinearity. Our design was implemented on an FPGA and was shown to achieve our claimed security by resisting leakage detection tests with 70 million traces.

A total area of 7849 Gate Equivalents is covered by our design. One S-box execution consumes 126 bits of randomness, which is largely required to achieve the bivariate security. We discussed the scaling of area, speed and randomness when increasing the order of security. Exactly how these costs scale for TI is our first direction for future research, while optimizing the randomness cost provides a second direction.

Acknowledgements

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). In addition, this work was partially supported by the Research Fund KU Leuven, OT/13/071, and by European Unions Horizon 2020 research and innovation programme under grant agreement No 644052 HECTOR. Begül Bilgin was partially supported by the FWO project G0B4213N. Oscar Reparaz is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). Thomas De Cnudde is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

1. Bilgin, B.: Threshold Implementations, As Countermeasure Against Higher-Order Differential Power Analysis. Ph.D. thesis, Univ. of Twente, Enschede (May 2015)
2. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A More Efficient AES Threshold Implementation. In: Pointcheval, D., Vergnaud, D. (eds.) Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8469, pp. 267–284. Springer (2014), `http://dx.doi.org/10.1007/978-3-319-06734-6_17`
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of

Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8874, pp. 326–343. Springer (2014), http://dx.doi.org/10.1007/978-3-662-45608-8_18
4. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-Offs for Threshold Implementations Illustrated on AES. IEEE Trans. on CAD of Integrated Circuits and Systems 34(7), 1188–1200 (2015), http://dx.doi.org/10.1109/TCAD.2015.2419623
5. Canright, D.: A Very Compact S-Box for AES. In: Rao and Sunar [22], pp. 441–455, http://dx.doi.org/10.1007/11545262_32
6. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M.J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999), http://dblp.uni-trier.de/db/conf/crypto/crypto99.html#ChariJRR99
7. Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) Methodology in Practice. International Cryptographic Module Conference (2013), http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002), http://dx.doi.org/10.1007/978-3-662-04722-4
9. De Cnudde, T., Bilgin, B., Reparaz, O., Nikova, S.: Higher-order Glitch Resistant Implementation of the PRESENT S-Box. In: Ors, B., Preneel, B. (eds.) BalkanCryptSec. Lecture Notes in Computer Science, vol. 9024. Springer-Verlag (2014)
10. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A Testing Methodology for Side-Channel Resistance Validation. NIST non-invasive attack testing workshop (2011), http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf
11. Goubin, L., Patarin, J.: DES and Differential Power Analysis (The "Duplication" Method). In: Ko, .K., Paar, C. (eds.) CHES. Lecture Notes in Computer Science, vol. 1717, pp. 158–172. Springer (1999), http://dblp.uni-trier.de/db/conf/ches/ches1999.html#GoubinP99
12. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=646764.703989
13. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: Rao and Sunar [22], pp. 157–171, http://dblp.uni-trier.de/db/conf/ches/ches2005.html#MangardPO05
14. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Ko, .K., Paar, C. (eds.) CHES. Lecture Notes in Computer Science, vol. 1965, pp. 238–251. Springer (2000), http://dblp.uni-trier.de/db/conf/ches/ches2000.html#Messerges00
15. Moradi, A., Mischke, O.: On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In: Bertoni, G., Coron, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8086, pp. 1–20. Springer (2013), http://dx.doi.org/10.1007/978-3-642-40349-1_1
16. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K.G. (ed.)

Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 69–88. Springer (2011), `http://dx.doi.org/10.1007/978-3-642-20465-4_6`

17. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4307, pp. 529–545. Springer (2006), `http://dx.doi.org/10.1007/11935308_38`
18. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. J. Cryptology 24(2), 292–321 (2011), `http://dx.doi.org/10.1007/s00145-010-9085-7`
19. Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In: in RSA Conference Cryptographers Track, ser. LNCS. pp. 192–207. Springer (2006)
20. Peeters, E., Standaert, F., Donckers, N., Quisquater, J.: Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In: Rao and Sunar [22], pp. 309–323, `http://dx.doi.org/10.1007/11545262_23`
21. Prouff, E., Roche, T.: Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 63–78. Springer (2011), `http://dx.doi.org/10.1007/978-3-642-23951-9_5`
22. Rao, J.R., Sunar, B. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3659. Springer (2005)
23. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: CRYPTO, LNCS, vol. 9215, pp. 1–20. Springer (2015)
24. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting Time Samples for Multivariate DPA Attacks. In: CHES, LNCS, vol. 7428, pp. 155–174. Springer (2012)
25. Rijmen, V.: Efficient Implementation of the Rijndael S-box, `http://www.researchgate.net/profile/Vincent_Rijmen/publication/2621085_Efficient_Implementation_of_the_Rijndael_S-box/links/0912f50f7a7be367d7000000?origin=publication_detail`
26. Schneider, T., Moradi, A.: Leakage assessment methodology - A clear roadmap for side-channel evaluations. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9293, pp. 495–513. Springer (2015), `http://dx.doi.org/10.1007/978-3-662-48324-4_25`
27. Standaert, F., Peeters, E., Quisquater, J.: On the Masking Countermeasure and Higher-Order Power Analysis Attacks. In: International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1, 4-6 April 2005, Las Vegas, Nevada, USA. pp. 562–567. IEEE Computer Society (2005), `http://dx.doi.org/10.1109/ITCC.2005.213`
28. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: Joye, M., Quisquater, J.J. (eds.) CHES. Lecture Notes in Computer Science, vol. 3156, pp. 1–15. Springer (2004), `http://dblp.uni-trier.de/db/conf/ches/ches2004.html#WaddleW04`