Glitches and Static Power Hand in Hand

Amir Moradi and Oliver Mischke

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

Abstract. Several masking schemes to protect cryptographic implementations against side-channel attacks have been proposed. A few considered the glitches, and provided security proofs in presence of such inherent phenomena happening in logic circuits. One which is based on multi-party computation protocols and utilizes Shamir's secret sharing scheme was presented at CHES 2011. It aims at providing security for hardware implementations – mainly of AES – against those sophisticated side-channel attacks that also take glitches into account. One part of this article deals with the practical issues and relevance of the aforementioned masking scheme. We first provide a guideline on how to implement the scheme for the simplest settings, and address some pitfalls in the scheme which prevent it to be practically realized. Solving the problems and constructing an exemplary design of the scheme, we provide practical side-channel evaluations based on a 65nm-technology Virtex-5 FPGA. We still observe univariate side-channel leakage, which is not expected according to the proven security of the scheme. We believe that the leakage is due to a combination of static power consumption and glitches in the circuit which is observed for the first time in practice. Dependency of static power consumption of nano-scale devices on processed data – which was warned before to be problematic – becomes now critical. Our result does not invalidate the given security proof of the scheme itself, but instead shows that the underlying model to obtain the proofs no longer fits to the reality. This is true not only for the scheme showcased here, but also for most other known masking schemes. As a result, due to the still ongoing technology shrinkage most of the available data-randomizing side-channel countermeasures will not be able to completely prevent univariate side-channel leakage of hardware implementations. Our work shows that new models must be created under which the security of new schemes can be proven considering leakages through both dynamic and static power consumption.

1 Introduction

With the increasing widespread of security-enabled embedded devices the protection of these devices against malicious users became of a greater concern. Even if these devices are protected by cryptographic algorithms which are very secure considering a black box scenario, with the discovery of side-channel attacks and especially power analysis in the late 90s [9], algorithms which are implemented without countermeasures can nowadays easily be broken. One of the reasons for this is that power analysis equipment is relatively cheap and already published attacks can be utilized by a moderately skilled attacker. This is especially bothersome since most of these devices must be considered as working in a hostile environment with easy access of an attacker, lowering the inhibition threshold to perform such an attack.

Different masking schemes, like boolean and multiplicative, have been proposed in order to randomize the intermediate computations and hence provide security against power analysis attacks. They indeed have been presented to the community in an arms race to counteract the also evolving new side-channel attacks. Most of these earlier masking schemes while considered secure under the used security model at that time, still exhibit a detectable univariate first-order leakage which is caused by glitches in the combinational circuits of hardware. For instance, we can mention the schemes presented in [18] and [5] which have later been shown to be vulnerable in [11] and [13] respectively. Taking these occurring glitches into account new masking schemes have been developed claiming glitch resistance. Threshold Implementation (TI) [15–17] is one of the more studied ones. It is based on a specific type of multi-party computation and applies boolean masking. However, making a correct implementation which fulfills all the requirements of TI is very challenging, and so far only the Noekeon and the PRESENT S-boxes could be successfully realized under its definitions [17, 19]. TI is supposed to be secure only against 1^{st}-order attacks, and accordingly it has been shown that it can be broken by a univariate mutual information analysis (MIA) [3, 17] or a 2^{nd}-order univariate collision attack [12].

Another recently proposed scheme [20], also based on multi-party computation protocols, utilizes the Shamir's secret sharing scheme [22] and claims security not only against 1^{st}-order attacks but also depending on the number of shares against higher-order multivariate ones.[1] One of our contributions in this paper is to address some flaws of this scheme and accordingly provide solutions, thereby allowing a practical realization of the scheme. In order to make an exemplary architecture of this scheme we have chosen a parameter set based on the minimum number of shares to supposedly provide protection against any univariate attack. While we doubt the practical relevance of the scheme because of the very high time and area overheads, more importantly we conduct practical side-channel experiments which disqualify the security claims in a real-world scenario. We show that our evaluation platform, a Xilinx Virtex-5 FPGA, still exhibits an exploitable univariate 2^{nd}-order leakage of our exemplary design.

In fact, the security proof given for our target scheme is sound, and our result does not show its shortcoming. On the contrary, it indicates the inconsistency of the underlying model with reality, which the security proof is based on. In most of the provably secure masking schemes only dynamic power consumption which is a result of the circuit activity is taken into account. With the continuation of downscaling in process technology the impact of static power gains dramatically. This is the important point that at the moment is not considered when proving

[1] A similar masking scheme using Shamir's secret sharing with a software platform as target has also been presented at CHES 2011 [7].

the security of a masking scheme, and indeed in our experiments this causes the defeat of the claimed protection. By using a combination of dynamic and static power consumption, which are inherently summed up by hardware, we give first practical evidence that the currently used model to prove the security of masking schemes is no longer valid when considering more advanced deep sub-micron process technologies.

2 Preliminaries

Before focusing on our target masking scheme, we specify the definition of different side-channel attacks w.r.t. their variate and the statistical moment applied. An attack which combines v different time instances – usually in v different clock cycles – of each power trace is called v-variate attack. Regardless of v the order of an attack is defined by the order of statistical moments which are considered in the attack. For instance, a CPA [4] which combines two points of each power trace by summing them up is a bivariate 1^{st}-order attack, and a CPA which applies the squared values of each trace is a univariate 2^{nd}-order attack. Those attacks where no specific statistical moment is applied, e.g., MIA [3], are distinguished only by v like univariate or bivariate MIA.

2.1 Target Scheme

Although the scheme presented in [20] is more or less general, we rewrite its basics for minimum settings and by considering the AES Rijndael as the target algorithm. By $\otimes$ we denote the multiplication in $\mathrm{GF}(2^8)$ using the Rijndael irreducible polynomial and by $\oplus$ the finite-field addition. The number of shares (and accordingly the number of Players) is fixed to 3 (i.e., degree of the underlying polynomial is 1, the most simplified setting in [20]). Regardless of the settings the scheme is expected to provide security against any univariate attacks.

Before starting the shared operations, one needs to select 3 distinct non-zero elements, so-called *public points*, $\alpha_1, \alpha_2, \alpha_3$ in $\mathrm{GF}(2^8)$. Moreover, it is required to precompute the first row $(\lambda_1, \lambda_2, \lambda_3)$ of the inverse of the Vandermonde (3×3)-matrix $(\alpha_i^j)_{1 \leq i,j \leq 3}$ as

$$\begin{aligned}
\lambda_1 &= \alpha_2 \otimes \alpha_3 \otimes (\alpha_1 \oplus \alpha_2)^{-1} \otimes (\alpha_1 \oplus \alpha_3)^{-1} \\
\lambda_2 &= \alpha_1 \otimes \alpha_3 \otimes (\alpha_1 \oplus \alpha_2)^{-1} \otimes (\alpha_2 \oplus \alpha_3)^{-1} \\
\lambda_3 &= \alpha_1 \otimes \alpha_2 \otimes (\alpha_1 \oplus \alpha_3)^{-1} \otimes (\alpha_2 \oplus \alpha_3)^{-1},
\end{aligned}$$

where x^{-1} denotes the multiplicative inverse of x in $\mathrm{GF}(2^8)$ using again the Rijndael irreducible polynomial. These elements, $\alpha_1, \alpha_2, \alpha_3$ and $\lambda_1, \lambda_2, \lambda_3$, are publicly available to all 3 Players.

Sharing a secret x is done by randomly selecting a secret coefficient a and computing 3 shares x_1, x_2, x_3 as

$$x_1 = x \oplus (a \otimes \alpha_1), \qquad x_2 = x \oplus (a \otimes \alpha_2), \qquad x_3 = x \oplus (a \otimes \alpha_3).$$

Each Player i gets only one share x_i without having any information about the other shares.

Reconstructing the secret x from the 3 shares x_1, x_2, x_3 can be done as

$$x = (x_1 \otimes \lambda_1) \oplus (x_2 \otimes \lambda_2) \oplus (x_3 \otimes \lambda_3).$$

Let us suppose a constant c and two secrets x and y which are represented each by 3 shares x_1, x_2, x_3 and y_1, y_2, y_3 constructed using the same public points $\alpha_1, \alpha_2, \alpha_3$ and by secret coefficients a and b respectively. In the following we consider the essential operations required for an AES S-box computation, and discuss about the role of each Player.

Addition with a constant, i.e., $z = c \oplus x$, in the shared mode can be done by each Player performing the addition as

$$\begin{array}{llcccc}
\text{Player 1}: & z_1 = x_1 \oplus c & = & x \oplus (a \otimes \alpha_1) \oplus c & = & (x \oplus c) \oplus (a \otimes \alpha_1) \\
\text{Player 2}: & z_2 = x_2 \oplus c & = & x \oplus (a \otimes \alpha_2) \oplus c & = & (x \oplus c) \oplus (a \otimes \alpha_2) \\
\text{Player 3}: & z_3 = x_3 \oplus c & = & x \oplus (a \otimes \alpha_3) \oplus c & = & (x \oplus c) \oplus (a \otimes \alpha_3).
\end{array}$$

Therefore, z_1, z_2, z_3 correctly provide the shared representation of z.

Multiplication with a constant, i.e., $z = c \otimes x,\ c \neq 0$, also can be performed in a similar way as

$$\begin{array}{llcccc}
\text{Player 1}: & z_1 = x_1 \otimes c & = & (x \oplus (a \otimes \alpha_1)) \otimes c & = & (x \otimes c) \oplus (a \otimes c \otimes \alpha_1) \\
\text{Player 2}: & z_2 = x_2 \otimes c & = & (x \oplus (a \otimes \alpha_2)) \otimes c & = & (x \otimes c) \oplus (a \otimes c \otimes \alpha_2) \\
\text{Player 3}: & z_3 = x_3 \otimes c & = & (x \oplus (a \otimes \alpha_3)) \otimes c & = & (x \otimes c) \oplus (a \otimes c \otimes \alpha_3),
\end{array}$$

and z_1, z_2, z_3 also provide the shared representation of z considering $a \otimes c$ as the secret coefficient.

Addition of two shared secrets, i.e., $z = x \oplus y$, is easily performed by

$$\begin{array}{ll}
\text{Player 1}: & z_1 = x_1 \oplus y_1 = x \oplus (a \otimes \alpha_1) \oplus y \oplus (b \otimes \alpha_1) = (x \oplus y) \oplus ((a \oplus b) \otimes \alpha_1) \\
\text{Player 2}: & z_2 = x_2 \oplus y_2 = x \oplus (a \otimes \alpha_2) \oplus y \oplus (b \otimes \alpha_2) = (x \oplus y) \oplus ((a \oplus b) \otimes \alpha_2) \\
\text{Player 3}: & z_3 = x_3 \oplus y_3 = x \oplus (a \otimes \alpha_3) \oplus y \oplus (b \otimes \alpha_3) = (x \oplus y) \oplus ((a \oplus b) \otimes \alpha_3).
\end{array}$$

z_1, z_2, z_3 provide the shared representation of z as well considering $a \oplus b$ as the secret coefficient.

Multiplication of two shared secrets, i.e., $z = x \otimes y$, is the challenging part. If each Player computes the multiplication of two shares as

$$\begin{aligned}
\text{Player 1}: \ & t_1 = x_1 \otimes y_1 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_1) \oplus (a \otimes b \otimes \alpha_1^2)\\
\text{Player 2}: \ & t_2 = x_2 \otimes y_2 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_2) \oplus (a \otimes b \otimes \alpha_2^2)\\
\text{Player 3}: \ & t_3 = x_3 \otimes y_3 = (x \otimes y) \oplus (((a \otimes y) \oplus (b \otimes x)) \otimes \alpha_3) \oplus (a \otimes b \otimes \alpha_3^2),
\end{aligned}$$

t_1, t_2, t_3 are not a correct shared representation of z because according to [20] the underlying polynomial is of a higher degree and does not have a uniform distribution. The solution given in [20] is as follows:

1. Each Player i after computing t_i, randomly selects a coefficient a_i, remasks t_i as

$$q_{i,1} = t_i \oplus (a_i \otimes \alpha_1), \qquad q_{i,2} = t_i \oplus (a_i \otimes \alpha_2), \qquad q_{i,3} = t_i \oplus (a_i \otimes \alpha_3),$$

   and sends each $q_{i,\forall j \neq i}$ to the corresponding Player j.
2. Now each Player i has three elements $q_{1,i}, q_{2,i}, q_{3,i}$, and reconstructs z_i as

$$z_i = (q_{1,i} \otimes \lambda_1) \oplus (q_{2,i} \otimes \lambda_2) \oplus (q_{3,i} \otimes \lambda_3).$$

Indeed, z_1, z_2, z_3 provide a correct shared representation of z considering $(a_1 \otimes \lambda_1) \oplus (a_2 \otimes \lambda_2) \oplus (a_3 \otimes \lambda_3)$ as the secret coefficient.

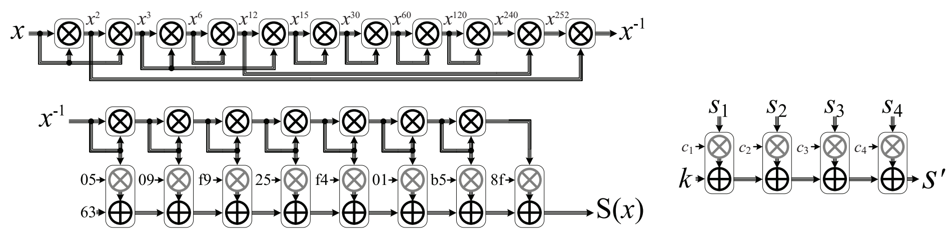
Square of a shared secret, i.e., $z = x^2$, cannot be computed in a straightforward way in contrast to what is stated in [20]. If each Player i squares its share x_i as

$$\begin{aligned}
\text{Player 1}: \quad & z_1 = {x_1}^2 & = \quad & x^2 \oplus (a^2 \otimes {\alpha_1}^2)\\
\text{Player 2}: \quad & z_2 = {x_2}^2 & = \quad & x^2 \oplus (a^2 \otimes {\alpha_2}^2)\\
\text{Player 3}: \quad & z_3 = {x_3}^2 & = \quad & x^2 \oplus (a^2 \otimes {\alpha_3}^2),
\end{aligned}$$

z_1, z_2, z_3 do not provide a correct shared representation of z unless – as also stated in [7] – the public points $\alpha_1, \alpha_2, \alpha_3$ as well as $\lambda_1, \lambda_2, \lambda_3$ are squared. If the result of squaring z_1, z_2, z_3 need to contribute in later computations where other secrets shared by original public points $\alpha_1, \alpha_2, \alpha_3$ are involved, z_1, z_2, z_3 must be remasked to provide a correct shared representation of z using the original public points. To do so a *FreshMasks* scheme is proposed in [7], but we consider the realization of squaring by giving the above mentioned shared multiplication algorithm the same shared secrets, i.e., $z = x \otimes x$. This, in fact, makes a correct representation of z using the desired unchanged public points.

In order to compute the inversion part of the AES S-box one can use the scheme presented in [21] as

$$x^{-1} = x^{254} = \Big(\big(x^2 \otimes x\big)^4 \otimes \big(x^2 \otimes x\big)\Big)^{16} \otimes \big(x^2 \otimes x\big)^4 \otimes x^2.$$



(a) S-box (b) MixColumns

Fig. 1. Block diagram of sequential operations necessary for an AES S-box and a forth of MixColumns

Since this scheme contains only a couple of square and multiply operations, using only the aforementioned shared multiplication algorithm the inversion part can be realized under our defined sharing settings. In contrast to what is claimed in both [20] and [7], the remaining part, i.e., the affine transformation, cannot be performed in a straightforward way. That is because – as also stated in [2] – the linear part of the affine transformation of the AES is a linear function over GF(2) not over GF(2^8). The solution for this problem is to represent the affine transformation over GF(2^8) and using the Rijndael irreducible polynomial. This actually has been presented before in [14] and [6] as

$$\begin{aligned}\text{Affine } (x) = \quad & 63 \; \oplus \; (05 \otimes x) \;\; \oplus (09 \otimes x^2) \; \oplus (\text{f9} \otimes x^4) \;\; \oplus (25 \otimes x^8) \oplus \\ & (\text{f4} \otimes x^{16}) \oplus (01 \otimes x^{32}) \oplus (\text{b5} \otimes x^{64}) \oplus (\text{8f} \otimes x^{128}) \quad .\end{aligned}$$

Therefore, by the diagram given in Fig. 1(a) we define the sequence of operations of a complete S-box computation considering the secret sharing restated above. Note that the modules denoted by black $\otimes$ indicate the shared multiplication, and those by gray $\otimes$ the multiplication with a constant.

3 Our Design

In order to implement the aforementioned scheme one needs to follow the requirements addressed in [20]. The goal of the scheme is to separate the side-channel leakage of the computations done by each Player in order to prevent any univariate leakage. As stated in [20] there are two possible ways to separate the leakage. Either the circuit of each Player is realized by dedicated hardware, e.g., one FPGA per Player, which does not seem to be practical, or the operations of each Player are separated in time. We follow the second option and have tried to mount the whole of the scheme in one FPGA – with the goal of a global minimum area-overhead – by the design shown in Fig. 2.

By means of a dedicated and carefully designed control unit we made sure that the Players sequentially get active. In other words, no computation or activity is done by the other Players when one Player is active. The design of the

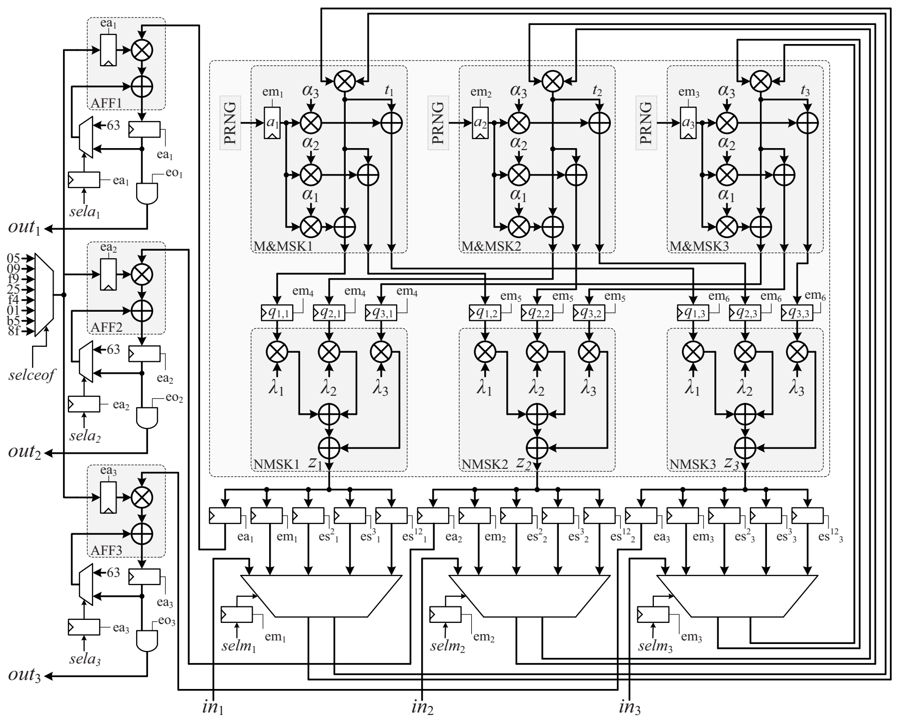


Fig. 2. Our design of the shared multiplication and addition to realize the AES S-box

shared multiplication module is slightly different to the other modules. In contrast to the others, where the computation on each share by the corresponding Player is independent of that of the other shares, the Players in the shared multiplication module need to communicate with each other. Therefore, we had to divide the computations of each share in this module into two parts by inserting a register between the two steps as explained in Section 2.1 (see registers marked by $q_{i,j}$ in Fig. 2).

Another important issue regarding our design is the way that the multiplexers are controlled. Since the shared multiplication module needs to get different inputs in order to realize a multiplication or a square, there should be a multiplexer to switch between different inputs. That is because – considering Fig. 1(a) – the shared multiplication module performs always squaring except in steps 2, 5, 10, and 11. Control signals which select the appropriate multiplexer input must be *hazardless*[2]. Otherwise, as an example, glitches on select signals of Player 1 while Player 2 is active will lead to concurrent side-channel leakage of two shares. Therefore, as a solution we provided some registers to control which input to be given to the target module.

[2] In the areas of digital logic a *dynamic hazard* means undesirable transient changes in the output as a result of a single input change.

For simplicity, we first explain how the shared multiplication module works:

- In the first clock cycle by activating enable signal em_1 the first share of both appropriate inputs are saved into their corresponding registers, get selected by select signal $selm_1$, and therefore are multiplied. At the same time the remasking process using a new random a_1 and public points $\alpha_1, \alpha_2, \alpha_3$ is performed. Note that the result of these computations are not saved in this clock cycle.
- The same procedure as in the first clock cycles is done on the second and the third shares one after each other in the second and the third clock cycles by activating enable signals em_2 and em_3 respectively.
- The results of the remasking for Player 1 (indeed provided by all 3 Players) which are available at the input of registers $q_{1,1}, q_{2,1}, q_{3,1}$ are stored at the forth clock cycle by enabling signal em_4. Therefore, the second step of the module gets active and performs the unmasking using $\lambda_1, \lambda_2, \lambda_3$ to provide the first share of the multiplication output. Note that again the result is not saved in this clock cycle.
- In the next two clock cycles (fifth and sixth) the same operation as the previous clock cycle is performed for Player 2 and Player 3 consecutively by enable signals em_5 and em_6.

Note that to save x^2, x^3, and x^{12} (see Fig. 1(a)) in the appropriate step, one of the signals $es^2_{i\in\{1,2,3\}}$, es^3_i, and es^{12}_i gets enabled at the same time with the corresponding em_i signal. In fact, we need six clock cycles to completely perform a shared multiplication or a square. It means that since we use only one shared multiplication module in our design, in $6 \times 11 = 66$ clock cycles the inverse of the given shared input is computed.

Afterwards in order to realize the affine transformation the multiplication-addition module (modules AFF1, AFF2, and AFF3 in Fig. 2) must also contribute into the computations. The Players in this module do not need to establish any communication and their computation is restricted to their own shares. Therefore, by appropriately selecting $sela_{i\in\{1,2,3\}}$ and enabling the ea_i signal the multiplication with constant and the shared addition both can be done in one clock cycle per share, i.e., three clock cycles in sum. Note that the same techniques as before to make hazardless control signals are used in the design of the multiplication-addition module. Also, the sequence of operations is similar to what is expressed for the first three clock cycles of the shared multiplication module. According to Fig. 1(a), during the affine transformation a multiplication-addition operation must be performed prior to each and after the last square. Therefore, after $3 \times 8 + 6 \times 7 = 66$ clock cycles the operations of an affine transformation is completed resulting in 132 clock cycles in sum to compute an S-box shared output.

One optimization option is to perform the multiplication-addition and the first three clock cycles of the squaring at the same time to save 24 clock cycles per S-box computation. According to the definition and the requirements of the scheme, it should not provide any security loss. However, since our main goal is to practically examine the side-channel leakage of this scheme, we ignored this

Table 1. Area and Time overhead of our design based on XC5VLX50 Virtex-5 FPGA (excluding state register, KeySchedule, PRNGs, initial masking, and final unmasking)

Design	**FF**		**LUT**		**Slice**		**SB**	**MC+ARK**	**Encryption**
	#	%	#	%	#	%	**CLK**	**CLK**	**CLK**
1 SB MC	315	1%	1387	5%	859	12%	2112	192	22 896
16 SB MC	4275	15%	21 328	74%	no fit		132	12	1431

optimization to be able to separately localize the side-channel leakage of each operation.

Though an optimized scenario to perform MixColumns is proposed in [20], by adding more multiplexer (and select register) to the multiplication-addition module our presented design can also realize MixColumns and AddRoundKey. This can be done according to the diagram given by Fig. 1(b) and selecting the appropriate coefficients c_1, c_2, c_3, c_4 corresponding to the rows of the matrix representation of MixColumns. After finishing all SubBytes transformations of one encryption round, i.e., $132 \times 16 = 2112$ clock cycles, every output byte of the MixColumns transformation in addition to the corresponding AddRoundKey can be computed in $3 \times 4 = 12$ clock cycles. That is, $12 \times 16 = 192$ clock cycles for whole of the MixColumns and AddRoundKey transformations. In sum, ignoring the required time for initial masking of the input and the key and for (pre)computing the round keys a whole encryption process takes $2112 \times 10 + 192 \times 9 + 3 \times 16 = 22\,896$ clock cycles.[3]

We should stress that – except the mentioned one – no time-optimization option exists for our single-S-box design since no more than one share is allowed to be processed at the same time. It is possible to reach a higher throughput by making multiple, e.g., 16, instances of our design inside the target FPGA and process all SubBytes and later all MixColumns in parallel. This, in fact, leads to a very high area-overhead (addressed by Table 1) that even cannot fit into the slices available in our target FPGA which is of the medium-size modern series. We should emphasize that the GF(2^8) multiplier we employed here is a highly optimized and pure combinational circuit, and the design is made for any arbitrary public values $\alpha_{i\in\{1,2,3\}}$ and λ_i.

4 Practical Evaluations

We used a SASEBO-GII [1] board as the evaluation platform. In order to realize the scheme we implemented our design on the Virtex-5 (XC5VLX50) FPGA embedded on the target board, and measured power consumption traces using a digital oscilloscope at the sampling rate of 1GS/s. A 1Ω resistor in the VDD path, a DC blocker, a passive probe, an amplifier, and restricting the bandwidth of the

[3] In the last round MixColumns is ignored and each separate AddRoundKey on one shared state value takes 3 clock cycles.

oscilloscope to 20MHz helped to obtain clear and low-noise measurements. All of our target designs run by a stable 3MHz oscillator during the measurements.

We made an exemplary design which performs only the initial AddRoundKey and SubBytes transformations on two given input bytes. We omitted the rest of the circuit in this design to focus only on the side-channel leakage caused during the S-box computation. The design gets two plaintext bytes $p^{(1)}$ and $p^{(2)}$, and makes three shares of each by means of the public points $\alpha_1, \alpha_2, \alpha_3$ and two separate random bytes. Two secret key bytes $k^{(1)}$ and $k^{(2)}$, which are fix inside the design, are similarly shared using two other separate random bytes. After XORing the corresponding shares of the plaintext and key bytes (AddRoundKey transformation) as $pk_i^{(j)} = p_i^{(j)} \oplus k_i^{(j)},\ j \in \{1,2\},\ i \in \{1,2,3\}$, the first three shares $pk_1^{(1)}, pk_2^{(1)}, pk_3^{(1)}$ are given to the S-box module. After 132 clock cycles – when the S-box shared output is ready – the second three shares $pk_1^{(2)}, pk_2^{(2)}, pk_3^{(2)}$ are provided as input of the same module. Finishing the second S-box computation, by means of $\lambda_1, \lambda_2, \lambda_3$ the results are unmasked for result validation.

We provided a clear trigger signal for the oscilloscope which indicates the start of the first and the end of the second S-box computation, thereby perfectly aligning the measured power traces. We also restricted the measurements to cover only the two S-box computations. In order to have the side-channel leakage of a similar but unprotected design as a reference, we made two other variants of our design. One is made by removing the intermediate $q_{i,j}$ registers of the shared multiplication module (see Fig. 2) and modifying the control unit; therefore, all three Players are active and perform the computation at the same time. Comparing the side-channel leakage of this variant to that of our original design can show the effectiveness of separating the computation of the Players. The second variant is constructed by simply turning off the PRNGs in our original design to keep all random bytes used for sharing and re-sharing at zero. Examining this variant we can localize the leakage points when evaluating our original design.

In the experiments shown below we selected the public points as $(\alpha_1, \alpha_2, \alpha_3) = (02, 03, 04)$ and accordingly $(\lambda_1, \lambda_2, \lambda_3) = (02, \mathrm{d2}, \mathrm{d1})$. We also kept the two secret key bytes fix and randomly selected the two input plaintext bytes. We start our evaluation by examining the first variant. Note that we modified the control unit in this version while still keeping it synchronized with the one of the original design. In other words, each shared multiplication is done in a single clock cycle and afterwards the circuit is idle for the next five clock cycles. The same holds for the multiplication-addition operation, i.e., all Players are active in one clock cycle and all off in the next two. In sum, it finishes one S-box computation in still 132 clock cycles. This is the reason for having low power consumption in a couple of adjacent clock cycles in an exemplary power trace of this variant shown by Fig. 3(a) where the sequence of operations are marked.

The technique we used to evaluate the side-channel leakage of our targeted designs is the *correlation-collision attack* [13]. It examines the leakage of one circuit instance that is used in different time instances. Therefore, it perfectly

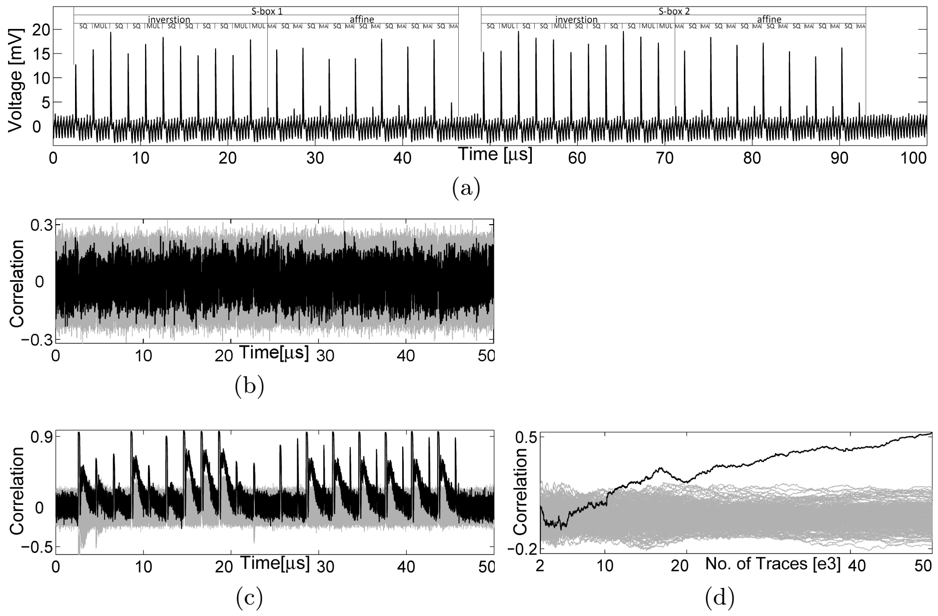


Fig. 3. First variant: evaluation results (a) a sample power trace, (b) first-order, and (c) second-order univariate attack result using 1 000 000 traces, (d) second-order over the number of traces

suits to our targeted designs since a single module is shared for both two S-box computations. This attack originally examines only the first-order leakage, but according to [12] it can be adopted to use higher-order moments and examine higher-order leakage. Unless otherwise stated, we concentrate on first- and second-order univariate leakage of our targets. As stated before, the goal of this scheme with minimum settings is to provide security against any univariate attack.

We collected 1 000 000 traces of the first variant and performed the aforementioned attack using the first- and second-order moments (averages and variances) targeting the linear difference between two used key bytes, i.e., $k^{(1)} \oplus k^{(2)}$. The results of the attack shown in Fig. 3(b) and Fig. 3(c) indicate no first-order but obvious second-order univariate leakage in the design which was expected. Also, Figure 3(d) shows the simplicity of recovering the second-order leakage (ca. 30 000 traces).

The second variant, where just the PRNGs are off, has lower power consumption compared to the first variant since the activity of each Player is restricted to one clock cycle and the glitches are controlled between the two steps of the shared multiplication module. A sample power trace of this variant is shown in Fig. 4(a). Since no randomness is contributed into the computations, the first-order leakage is highly expected. It is indeed confirmed by the result of the

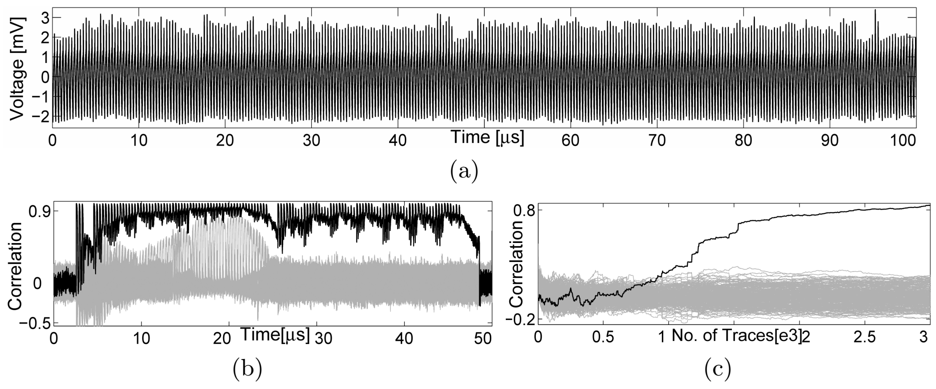


Fig. 4. Second variant: evaluation results (a) a sample power trace, (b) first-order univariate attack result using 10 000 traces, and (c) over the number of traces

same attack as before using 10 000 measurements in Fig. 4(b) (see also Fig. 4(c) indicating ca. 2 000 as the number of required traces).

Coming back to our original design, the power consumption traces are similar to that of the second variant with slightly higher amplitude due to the random numbers. Having 1 000 000 measurements of the design we performed the same attack as before (using the first-order moments) which – as expected – led to an unsuccessful result (depicted by Fig. 5(a)). As can be seen in Fig. 5(b), the same attack using the second-order moment is surprisingly successful. Please note that it means a univariate MIA [3] with a suitable model can also be successful.

5 Discussions

In order to localize the source of this univariate leakage we made and evaluated several exemplary designs. We confirmed that at each clock cycle not more than one Player is active and no computations on more than one share is performed. We believe and provide practical evidence supporting the assumption that the source of this leakage is a combination of static and dynamic power consumption. The dependency of static power consumption on the input and output of the logic gates has been investigated before. For instance, this issue has been pointed out in [10], where static power consumption of sub-90nm gates in the simulation domain is examined. Indeed, this issue was not taken as a serious threat by the community, and our results which are based on a Virtex-5 FPGA employing a 65nm technology shows the practical relevance of this problem.

According to the diagram of our design (Fig. 2), the registers and their following combinational circuits contain the result of computations of each Player. Therefore, the static power consumption of each sub-circuit related to each Player depends on data (charges) available and stable in the registers and combinational circuits. However, the static power consumption of the design is the sum

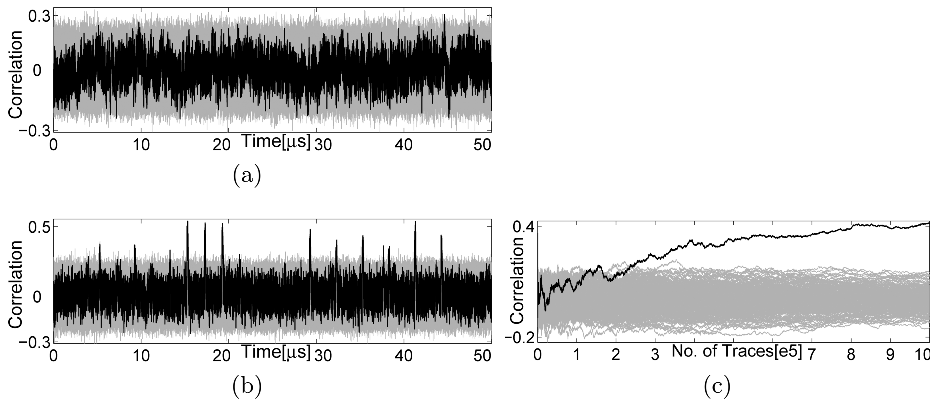


Fig. 5. Original design: evaluation results using 1 000 000 traces (a) first-order univariate attack result, (b) second-order univariate attack result, and (c) over the number of traces

of the aforementioned share-dependent static powers, and therefore depends on unshared data (please see [23] for more information about the possible attacks when the leakage of the shares are summed up).

As an evidence of our claim we refer to the attack results of both variants of our design. In the first variant (Fig. 3(c)) correlation for the correct guess is clearly high not only at the clock cycle in which the computation takes place but also in the next few clock cycles when the circuit is idle. We should stress that the correlation value does not stay high during whole of the e.g., 5, idle clock cycles because the DC blocker – which we used in our measurement setup – slowly cancels the influence of the static power consumption. Note that the target FPGA was clocked at a frequency of 3MHz, by which we made sure that the dynamic power consumption of adjacent clock cycles do not overlap. The same holds for the second variant (Fig. 4(b)); correlation for the correct guess does appear high not only on the peak of the clock cycles. On the contrary, it shows the dependency of the power consumption on processed (and later saved) data even between two consecutive clock cycles. Indeed, we showed the attack results in Fig. 3(c) and Fig. 4(b) using much more traces than required in order to clearly show the appearance of side-channel leakage through static power consumption.

In order to provide more convincing evidences we repeated our experiments on the second variant by down-clocking the target FPGA to an extremely slow frequency of 1500Hz to keep the dynamic power consumption of two consecutive clock cycles as far away as possible thereby observing the static power consumption effect clearly. We also repeated this experiments using a SASEBO board using a Virtex-II pro FPGA employing a 90nm transistor technology. The results of all these experiments which confirm our claim are given in the Appendix.

As a result, in our original design when the computation of each share is separated, still the total static power consumption is data dependent. We examined our original design for several different public points $\alpha_1, \alpha_2, \alpha_3$. In all cases the leakage appeared at the third clock cycle of the shared multiplication module (when em_3 – see Fig. 2 – gets active) and in some rare cases at the second clock cycle as well. However, it never showed the leakage in the first clock cycle. It indeed confirms our claim that the leakage of one or two shares are available through static power consumption and are combined with the leakage of the next share through dynamic power consumption. Furthermore, for some public points we had to collect much more traces, e.g., 10 000 000, to clearly see the leakage. Different public points actually have different impact on the remasking part of the shared multiplication module and hence on its leakage. We should emphasize that we made several other designs to avoid this leakage by changing the input of the combinational circuit to zero after finishing their computation. However, all of our efforts failed since the result of the computations must be stored in a register which itself affects the static power consumption.

6 Conclusions

In this work we have demonstrated how to correctly implement a provably-secure glitch-resistant masking scheme based on Shamir's secret sharing and multi-party computation protocols [20]. By making certain that in each point in time only operations on a single share are performed, there should in theory exist no exploitable univariate leakage. While we could show its opposite in practice, this does not invalidate the security proofs of this scheme because the model used in proofs takes only dynamic power consumption, e.g., by glitches, into account. We, on the other hand, were able to extract the secrets by using a combination of dynamic and static power consumption which is out of the scope of the proofs. This indeed is the first practical evidence that static power consumption can also thwart a sophisticated countermeasure.

We demonstrated that the threat of leakage by static power consumption can no longer be ignored. Since the semiconductor technology is still shrinking, e.g., 45nm, 40nm, and 28nm in Spartan-6, Virtex-6, and 7 series FPGAs of Xilinx, the problem observed in this article becomes even more critical not only for the underlying scheme of this article but also for other masking schemes, e.g., threshold implementation [17] and private circuits [8]. In general our findings in this work may cloud the future of side-channel countermeasures which are based on data randomization.

References

1. Side-channel Attack Standard Evaluation Board (SASEBO). Further information are available via `http://www.rcis.aist.go.jp/special/SASEBO/index-en.html`.

2. Error in Report 2011/516: Protecting AES with Shamir's Secret Sharing Scheme by Louis Goubin and Ange Martinelli. Discussion forum of ePrint Archive: Report 2011/516 `http://eprint.iacr.org/forum/read.php?11,549,549#msg-549`, Sep 2011.
3. L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon. Mutual Information Analysis: a Comprehensive Study. *J. Cryptology*, 24(2):269–291, 2011.
4. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
5. D. Canright and L. Batina. A Very Compact "Perfectly Masked" S-Box for AES. In *ACNS 2008*, volume 5037 of *LNCS*, pages 446–459. Springer, 2008. the corrected version at Cryptology ePrint Archive, Report 2009/011 `http://eprint.iacr.org/`.
6. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
7. L. Goubin and A. Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In *CHES 2011*, volume 6917 of *LNCS*, pages 79–94. Springer, 2011.
8. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LLNCS*, pages 463–481. Springer, 2003.
9. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
10. L. Lin and W. Burleson. Leakage-based differential power analysis (LDPA) on sub-90nm CMOS cryptosystems. In *ISCAS 2008*, pages 252–255. IEEE, 2008.
11. S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005*, volume 3659 of *LNCS*, pages 157–171. Springer, 2005.
12. A. Moradi. Statistical Tools Flavor Side-Channel Collision Attacks. In *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 428–445. Springer, 2012.
13. A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010.
14. S. Murphy and M. J. B. Robshaw. Essential Algebraic Structure within the AES. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 1–16. Springer, 2002.
15. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.
16. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementations of Non-Linear Functions in the Presence of Glitches. In *ICISC 2008*, volume 5461 of *LNCS*, pages 218–234. Springer, 2008.
17. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.
18. E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In *FSE 2005*, volume 3557 of *LNCS*, pages 413–423. Springer, 2005.
19. A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *J. Cryptology*, 24(2):322–345, 2011.
20. E. Prouff and T. Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In *CHES 2011*, volume 6917 of *LNCS*, pages 63–78. Springer, 2011.

21. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
22. A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.
23. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 112–129. Springer, 2010.

Appendix

We reduced the clock frequency of the target FPGA to 1500Hz in order to limit the effect of dynamic power consumption at each clock cycle of the power traces and mainly examine the static power consumption. We also implemented the second variant of the design on a SASEBO board which employs a Virtex-II pro FPGA. Since it is using a 90nm transistor technology, we expect to see the static power consumption as well. Moreover, using the SASEBO platform we could obtain the traces with a lower amount of electrical noise compared to SASEBO-GII due to their different PCB design and technology.

In the first step we kept the same measurement setup as stated in Section 4, i.e., measuring at the VDD path using a DC blocker and a passive probe, but we decreased the sampling rate to 100kS/s to avoid dealing with very long traces. This indeed does not affect on our goal which is to examine the static power available between consecutive clock cycles. We measured 100 000 traces of the second variant implemented on the Virtex-II pro FPGA and mounted the same attack as illustrated in Section 4 using the first-order moment. The attack result is shown by Fig. 6 indicating no success in observing any dependency between the static power consumption and the saved data since the correlation value for the correct guess appears high only at the clock cycle peaks.

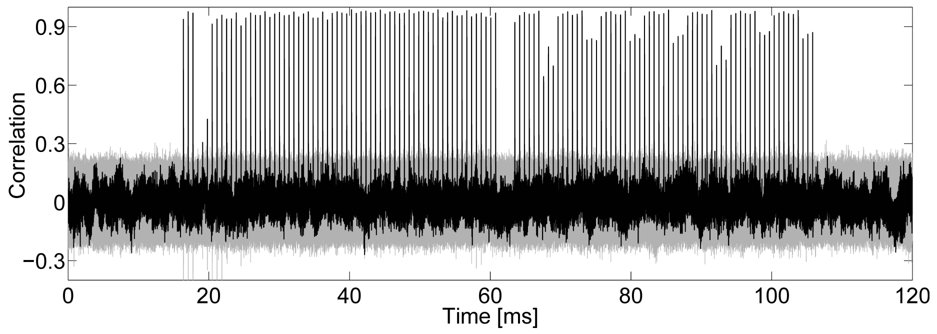


Fig. 6. Second variant on SASEBO (Virtex-II), 1500Hz, DC blocker: first-order univariate attack result using 100 000 traces

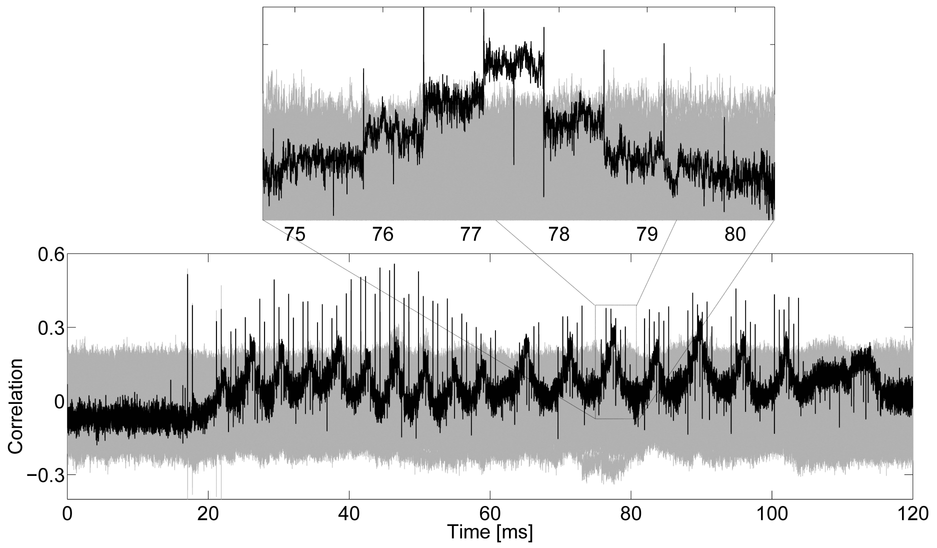


Fig. 7. Second variant on SASEBO (Virtex-II), 1500Hz, GND path, passive probe, DC50 mode: first-order univariate attack result using 100 000 traces

As stated before, we employed a DC blocker to reduce the electrical noise and obtain clear traces. However, it can remove the effect of the static power consumption as well if two consecutive clock cycles are far away from each other. Therefore, we removed the DC blocker and repeated the same experiment as before on traces measured from the GND path using a passive probe and setting the oscilloscope to DC50 ohm mode. The result of the corresponding attack depicted in Fig. 7 clearly shows that correlation values for the correct guess are high not only at the clock edges but also in the time interval between some consecutive clock cycles. This indeed confirms our claim that the information leakage through static power consumption is practically detectable using a suitable measurement setup. We should stress that the attack is not as effective as the previous experiment. The reason, as stated before, is due to the much better and clearer traces that we could collect using the DC blocker.

We also repeated both experiments on the Virtex-5 FPGA of the SASEBO-GII platform. The result of the first experiment – using the DC blocker – is shown by Fig. 8, and similar to the Virtex-II case it does not provide any opportunity to see the effect of the static power consumption. However, removing the DC blocker and measuring at the VDD path using a differential probe led to the result shown by Fig. 9. In this case, the traces were much more noisy than in all previous experiments which has a strongly negative effect on the attack efficiency. However, the influence of the static power consumption on the traces can be observed in the attack results (Fig. 9). We should emphasize that it was not possible to do the measurements at the GND path of SASEBO-GII. We did a couple of tries, but the target FPGA stops to be functional due to a lack of

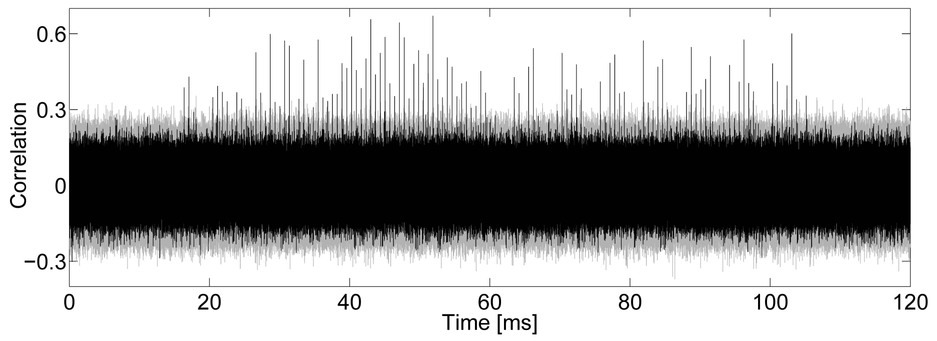


Fig. 8. Second variant on SASEBO-GII (Virtex-5), 1500Hz, DC blocker: first-order univariate attack result using 100 000 traces

current. This is because the resistor in the GND path affects the current of all VDD lines, i.e., VDD_INT, VDD_IO, VDD_AUX. However, since the resistor in the VDD path is only affecting VDD_INT of the target FPGA, enough current can be provided when measuring at the VDD path. All these problems are due to the design of the SASEBO-GII platform.

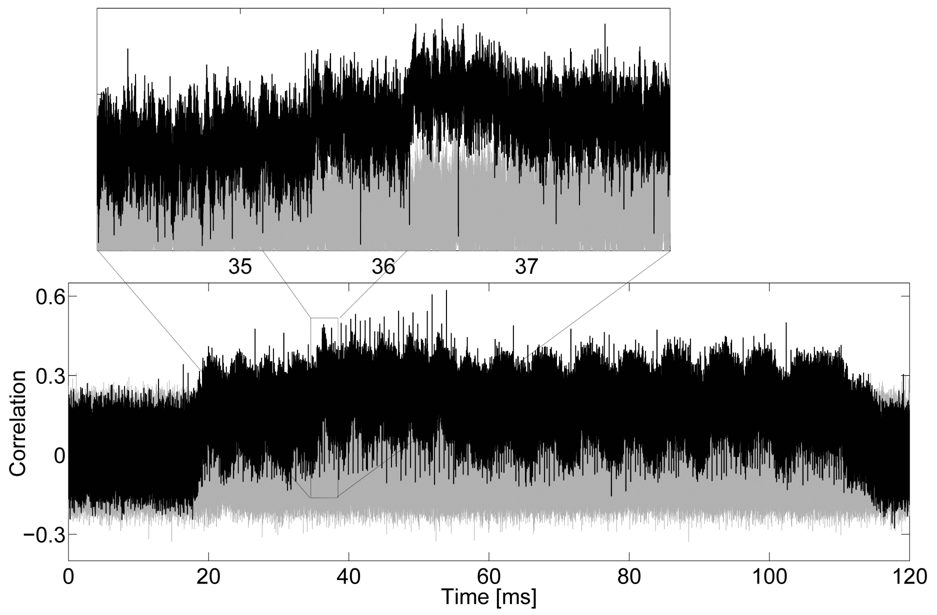


Fig. 9. Second variant on SASEBO-GII (Virtex-5), 1500Hz, VDD path, differential probe: first-order univariate attack result using 100 000 traces