Detecting Hidden Leakages

Amir Moradi[1], Sylvain Guilley[2,3], Annelie Heuser[2⋆]

[1] Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
amir.moradi@rub.de
[2] TELECOM-ParisTech, Crypto Group (COMELEC dpt), Paris, France
firstname.lastname@telecom-paristech.fr
[3] Secure-IC S.A.S., Rennes, France

Abstract. Reducing the entropy of the mask is a technique which has been proposed to mitigate the high performance overhead of masked software implementations of symmetric block ciphers. Rotating S-box Masking (RSM) is an example of such schemes applied to AES with the purpose of maintaining the security at least against univariate first-order side-channel attacks. This article examines the vulnerability of a realization of such technique using the side-channel measurements publicly available through DPA contest V4. Our analyses which focus on exploiting the first-order leakage of the implementation discover a couple of potential attacks which can recover the secret key. Indeed the leakage we exploit is due to a design mistake as well as the characteristics of the implementation platform, none of which has been considered during the design of the countermeasure (implemented in naive `C` code).

Keywords: Side-channel analysis, leakage detection, variance test, NICV, correlation-collision, CPA, hidden models, linear regression.

1 Introduction

Counteracting side-channel analysis attacks, as a major concern for embedded cryptographic solutions, is a must for today's both software- and hardware-based applications. One of the most studied countermeasures is masking [9,12,25,35], which by randomizing the secret internals aims at cutting the relation between the side-channel leakages and predictable processes. Realization of masking in hardware platforms faces many challenges due to the uncontrolled glitches happening inside the masked circuits. Since these issues are out of scope of this article, the interested reader is referred to [26,29,32,38]. Although there exist a couple of different masking techniques, the focus of this work is on Boolean masking for software-based platforms, where the challenges are mainly due to their significant overhead.

If masking is correctly realized in software, it can significantly increase the complexity of a successful attack. The goal of most of the techniques is to prove

⋆ Annelie Heuser is a Google European fellow in the field of privacy and is partially founded by this fellowship.

the impossibility of first-order attacks if the implementation as well as the leakage models follow the corresponding assumptions. As stated, the significant overhead needed to realize the masking schemes is amongst their major drawbacks. This overhead is due to two main issues:

- processing the mask and the masked data. For example, the linear operations (e.g., MixColumns of AES) must be performed on the masked data as well as on the mask, and
- on-the-fly recomputation of the masked look-up tables which are responsible to realize the non-linear operations, e.g., AES S-box.

The first issue is usually not the most dominant part and stays as is for most of the masking schemes. However, many solutions have been proposed to relax the second problem. Some focused on avoiding look-up tables for non-linear operations (like with secure multiplication — the interested reader is referred to [39,17]). It is worth to mention that masking schemes usually assume uniformly distributed random masks. Therefore, on-the-fly recomputation of the S-box is unavoidable unless a huge memory is available to precompute all the necessary tables [36]. Since dealing with this overhead is challenging, a couple of heuristic scenarios, e.g., reusing the mask for certain S-boxes, have been used mainly by industry sector[1]. However, each of these heuristics has a drawback which may lead to a seriously vulnerable implementation (see [10]). Instead, reducing the entropy of the mask is the idea followed by [30,31]. Use of fewer mask values allows precomputing all masked look-up tables and fit them to the small-size platforms, e.g., smartcards or microcontrollers with a few Kilobytes of flash memory. These schemes claim to provide the first-order security, which is defined as follows:

> For all possible mask values, mean of the side-channel leakages based on the predictable secret internals, e.g., one S-box output, is independent of the selected internal, i.e., $\mathbb{E}(l|v)$ is constant, where l and v denote leakage and a secret internal respectively.

In this work we mainly concentrate on a software implementation of Rotating S-boxes Masking (RSM) as a low-entropy masking scheme for AES [31]. We first in Section 2 restate the scheme and provide the necessary notations for formal discussions. Next we focus on an implementation which is publicly available through the *DPA Contest V4* [43]. We also use the corresponding side-channel measurements (of DPA Contest V4) to perform our security evaluations. The practical analyses, which are given in Section 3, aim at examining the existence of a first-order leakage. Two attacks are detailed: one correlation-collision (without a model) and then one correlation attack (with a model). Despite the claims of the original scheme as well as the security proofs, our analysis exploits a strong first-order leakage allowing us to recover the first 128-bit round key using less than 200 measurements. We also provide theoretical reasoning behind the exploitable leakage as well as a solution to prevent it in Section 4.

[1] based on the authors' observations

As related works we should address three recently published articles [3,23,46] which made use of DPA Contest V4 measurements. Although all of these articles provide many useful discussions and analysis tools, none of them exploits the first-order leakage that we present here. We give more detailed comparison between these works and our contribution in Section 4.

2 Masking in Software

Cryptographic software can be protected against differential side-channel attacks by having the sensitive intermediate variables depend on some random numbers. This strategy is called masking. The procedure consists in splitting every sensitive intermediate variable into several shares randomly, with the property that there exists a way to constructively recombine them to recover the sensitive variable. A classical sharing is the first-order Boolean additive masking, where a sensitive variable X is split in two shares S_0 and S_1 in such a way $X = S_0 \oplus S_1$. Typically, in this scheme, S_1 can be drawn *uniformly* randomly, and will be called the mask. Then S_0 is computed as $X \oplus S_1$. It is well known that any linear operation l is easy to evaluate in this paradigm: it is indeed sufficient to compute l on each share individually. The reason is that $S'_0 = l(S_0)$ and $S'_1 = l(S_1)$ is a sharing of $l(X)$. However, this does not apply to non-linear operations, such as the computation of a substitution box (S-box). Most of the research effort in the field of masking has thus been spent on this topic.

As we shall detail in the sequel, variants of masking schemes have been put forward. Their motivations are manifold:

- *first-order* masking might not be secure enough, i.e., more shares are required;
- some cryptographic functions are not *Boolean* (e.g., RSA is based on modular arithmetic, hence is preferably masked in some ring $\mathbb{Z}_N$);
- there are situations where the mask cannot be injected *additively* (which has the merit of being compatible with the key addition stage), but rather multiplicatively [19] or via a homographic function [14].

2.1 Traditional Scheme

Historically, the initial masking strategy was called the "S-box precomputation". We illustrate in the sequel such masking on a substitution permutation network (SPN) such as the AES, where the S-box is called $\mathsf{SubBytes}$. For each unique S-box table in the design (e.g., one in case of AES and eight in case of DES), two random variables S_1 and S'_1 are drawn, in order to mask respectively its input and its output. Using them, a so-called masked S-box $\mathsf{MaskedSubBytes}$ is computed [27]. For every input Y, $\mathsf{MaskedSubBytes}$ evaluates $\mathsf{MaskedSubBytes}(Y) = \mathsf{SubBytes}(Y \oplus S_1) \oplus S'_1$. This table can be used to securely traverse the S-box of the first share $S_0 = X \oplus S_1$, since $\mathsf{MaskedSubBytes}(S_0)$ is equal to $\mathsf{SubBytes}(X) \oplus S'_1$, and thus this new share combined with S'_1 is a

valid sharing of $\mathsf{SubBytes}(X)$. Nevertheless, in this operation, neither X nor $\mathsf{SubBytes}(X)$ appears unmasked, hence the security. So to protect a complete algorithm, the procedure is as follows; each plaintext byte P is first masked with S_1. Then, usually, the first operation is the addition with a key k. This operation is *on purpose* compatible with the masking, meaning that it is secure to add k to $P \oplus S_1$: indeed, it yields $(P \oplus k) \oplus S_1$, which together with S_1 is a sharing of $P \oplus k$. The share $(P \oplus k) \oplus S_1$ can now enter the precomputed $\mathsf{MaskedSubBytes}$ as already discussed. The computation goes on this way until the end of the algorithm, where it is eventually secure to demask the masked ciphertext with S'_1 (after the last key addition).

Such a protection is especially efficient for algorithms such as AES that uses several instances of the same S-box. Indeed, the precomputation of $\mathsf{MaskedSubBytes}$ (which will consist in 2×256 XORs and 256 copies of bytes) is factored for each invocation of the S-box (16 times per round, for all the 14 rounds in the case of AES-256). Since MixColumns is linear and hence transparent to Boolean masking, it should generally be performed on both shares. However, since $\text{MixColumns}(S'_1, S'_1, S'_1, S'_1) = S'_1$, it is sufficient to perform MixColumns on only one share.

2.2 Problems

The "S-box precomputation" has two types of contradictory drawbacks:

1. First of all, it has an inherently low security level, because some efficient attacks have reported, such as second-order attacks (that combine two leaking samples in a view to remove their common mask) are very practical. Moreover, there exist efficient techniques, e.g., [34,44], which target the precomputation of $\mathsf{MaskedSubBytes}$.
2. Second, it is already costly in practice, both in terms of cycle count (owing to the long S-box precomputation), and in terms of mask "entropy" budget. We recall that producing random numbers is difficult and costly; indeed, in theory, the modelization of masking requires independent and uniformly distributed masks. Even if masks are practically produced by an algorithmic pseudo-random generator (e.g., a stream cipher), this operation is obviously consuming resources.

Therefore researches have been carried out in these two directions. Without surprise, increasing the number of shares does impact negatively the performances. At the opposite, it is interesting to note that some simplifications successfully managed to maintain a notion of security while avoiding the precomputation stage and reducing the required pool of entropy.

2.3 Multi-Mask FEMS vs Mono-Mask LEMS

Systematic countermeasures aim at fixing the problem of the masks reuse. Indeed, this can be exploited by a combination of the two leakage samples using

the same mask to cancel or bias it. If the two leaking operations are similar, e.g., two computations of S-box, then the attack is referred to as a *collision attack* [6]. Otherwise, the attack is generally termed bivariate, and can consist in second-order CPA [28], multivariate MIA [18], or any other variant (e.g., [15]). For this reason, every intermediate variable is masked independently (e.g., the same masked S-box cannot be used twice), and the sharing is done with strictly more than two shares. Hence the name multi-mask fully entropic masking scheme (FEMS, as coined in [46]). However, this generalization is not trivial. For example, the first attempt to adapt the *precomputation S-box* scheme to d masks (i.e., $d+1$ shares) by [41] happened to be flawed. Indeed, a dependence with the sensitive variable could be exhibited by combining only two shares [13]. The design error was that only one masked table was used. A repaired version has been presented recently at [11]; it employs $d+1$ tables that need each to be precomputed d times (hence a quadratic complexity overhead in the number of masks). Other provably secure schemes have been promoted, such as the computation of the S-box in a Galois field; refer for instance to [39]. In some contexts (e.g., AES), it is the most efficient scheme, but still with roughly d^2 complexity.

At the opposite direction, some masking schemes have been designed to limit the amount of entropy. They are referred to as LEMS (low-entropy masking schemes) in [46]. Specifically, the masking scheme requires only one mask, that can take only a small number of values. This allows to precompute once for all possible masked S-boxes, and to store them *hardwired* in memory. This strategy is winning in terms of performance (albeit at the expense of more ROM). Security-wise, as the mask S_1 is no longer uniformly distributed, zero-offset [45] or mutual information attacks [2] become possible. But the degree to which the leakage shall be raised for a CPA attack – on a platform with a linear leakage function – to be successful can be made strictly larger than three (see next section), and thus becomes the relevant security parameter.

2.4 RSM

Rotating S-box Masking (RSM) is an example of such low entropy masking schemes [31]. The mask values are chosen in such a way that the leakage caused by the masked variable $X \oplus S_1$ depends on X only at degree 4. It is explained in [4] that such security can be reached if the masks are distributed as the 16 codewords of the $[8, 4, 4]$ linear code, extension with one parity bit of the $[7, 4, 3]$ Hamming code.

3 Practical Realization

3.1 DPA Contest V4

How the Scheme is Implemented The investigated cipher is AES-256 in encryption (Electronic Code Book) mode. It complies with the NIST FIPS standard [33]. In the notations that will follow, some minor adjustments are done

with respect to the standard; for instance, depending on the context, SubBytes (resp. MixColumns) can be considered on the whole state or on individual bytes (resp. individual columns). It is mainly coded in the `C` language, and is compiled by `avr-gcc`; only some constants to be stored in Flash memory are given in an assembly code. The implementation realizing the RSM scheme is supposed to provide security against *univariate* side-channel attacks up to order 3 if the leakage model is linear.

It can be considered that the protection by masking is added on top of an unprotected AES. This "base AES" has those features:

- The key schedule is precomputed.
- The sixteen substitution boxes (S-boxes) are called in this order:

```
0, 2, 4, 6, 8, 10, 12, 14,   // Even S-boxes first
1, 3, 5, 7, 9, 11, 13, 15.   // Odd S-boxes second
```

- The MixColumns operation is computed on a byte-by-byte basis, using an xtime table.

The masking protection is an additive Boolean masking scheme, with statically masked S-boxes (as introduced in Section 2.4). It adds to the "base AES" those features:

- Sixteen values of the mask (noted S_1 in Section 2), are noted as M_i, $i = [\![0, 15]\!]$. Those values are incorporated in the computation. They constitute a space vector, defined as {`0x00`, `0x0f`, `0x36`, `0x39`, `0x53`, `0x5c`, `0x65`, `0x6a`, `0x95`, `0x9a`, `0xa3`, `0xac`, `0xc6`, `0xc9`, `0xf0`, `0xff`}, and are public information. They are precomputed as state-wide masks, called $\mathsf{Mask}_{\mathsf{offset}}$ and defined as:

$$\begin{aligned}\mathsf{Mask}_{\mathsf{offset}} = ((&M_{\mathsf{offset}+0},\ M_{\mathsf{offset}+1},\ M_{\mathsf{offset}+2},\ M_{\mathsf{offset}+3}),\\ &(M_{\mathsf{offset}+4},\ M_{\mathsf{offset}+5},\ M_{\mathsf{offset}+6},\ M_{\mathsf{offset}+7}),\\ &(M_{\mathsf{offset}+8},\ M_{\mathsf{offset}+9},\ M_{\mathsf{offset}+10}, M_{\mathsf{offset}+11}),\\ &(M_{\mathsf{offset}+12}, M_{\mathsf{offset}+13}, M_{\mathsf{offset}+14}, M_{\mathsf{offset}+15}))\ .\end{aligned}$$

  Notice that in the equation above, the layout of the bytes is *transposed* with respect to the canonical representation of the state (i.e., lines represent columns).
- A random offset, noted offset, is drawn randomly in $[\![0, 15]\!]$ at the beginning of the computation; it determines the allocation of the masks for each byte of the state. Explicitly, the state byte i is masked by mask $M_{\mathsf{offset}+i}$. In this equation, $\mathsf{offset} + i$ is to be understood "modulo 16". We do the same assumption in the sequel concerning indices of bytes in a state.
- The S-box is replaced by sixteen masked S-boxes, that are stored precomputed; their equation is $\mathsf{MaskedSubBytes}_i(X) = \mathsf{SubBytes}(X \oplus M_i) \oplus M_{i+1}$, where X is a byte. This means that the output mask of each S-box is the *successor* of the input mask. This also explains why S-boxes are not called in the natural order; the goal is to prevent unfortunate demasking that might occur otherwise.

– To pass through the linear layer, the mask bytes are *compensated* (by exclusive-or), thanks to sixteen 128-bit precomputed constants, that are equal to:

$$\begin{aligned}&\mathsf{MaskCompensation}_{\mathsf{offset}} = \mathsf{Mask}_{\mathsf{offset}} \oplus \mathsf{MixColumns}(\mathsf{ShiftRows}(\mathsf{Mask}_{\mathsf{offset}}))\\ &= \mathsf{Mask}_{\mathsf{offset}} \oplus (\\ &\quad \mathsf{MixColumns}(M_{\mathsf{offset}+0},\ M_{\mathsf{offset}+5},\ M_{\mathsf{offset}+10}, M_{\mathsf{offset}+15}),\\ &\quad \mathsf{MixColumns}(M_{\mathsf{offset}+4},\ M_{\mathsf{offset}+9},\ M_{\mathsf{offset}+14}, M_{\mathsf{offset}+3}),\\ &\quad \mathsf{MixColumns}(M_{\mathsf{offset}+8},\ M_{\mathsf{offset}+13}, M_{\mathsf{offset}+2},\ M_{\mathsf{offset}+7}),\\ &\quad \mathsf{MixColumns}(M_{\mathsf{offset}+12}, M_{\mathsf{offset}+1},\ M_{\mathsf{offset}+6},\ M_{\mathsf{offset}+11}))\ .\end{aligned}$$

  This operation can be termed a "trans-masking", insofar as it simultaneously removes the mask used to protect the linear part of the current round and remasks with the new mask suitable for the S-boxes at the next round, and so without revealing any sensitive variable unmasked.
– For the last round, the compensation is slightly different, because there is no MixColumns. Instead of $\mathsf{MaskCompensation}_{\mathsf{offset}}$, the following constant is added by exclusive-or to the state to remove the mask and generate the ciphertext:

$$\mathsf{MaskCompensationLastRound}_{\mathsf{offset}} = \mathsf{ShiftRows}(\mathsf{Mask}_{\mathsf{offset}})\ .$$

The protected AES can thus be represented by the algorithm 1. The unprotected version of this algorithm can be recovered by erasing the lines in blue, and by trading MaskedSubBytes for SubBytes. This algorithm runs in constant time (the test at line 10 does not depend either on plaintext or roundkeys), so timing attacks [16] do not apply.

How the Measurement is Performed The information related to experimental setup is as mentioned on the DPA contest V4 website [43]. The whole design is loaded into an ATMega163 8-bit smartcard, and evaluated on a SASEBO-W platform. The measurements were taken using a LeCroy wave-runner 6100A oscilloscope by means of a Langer EMV 0–3 GHz EM probe. The acquisition bandwidth is 200 MHz and the sampling rate $F_S = 500$ MS/s. The smartcard is powered at 2.5 V and clocked at 3.57 MHz by the on-board Xilinx Spartan-6 FPGA.

3.2 Analysis

Before doing the analysis it is worth to have a look at the mean of the traces and specify the operations performed at different time periods. Figure 1 shows a mean trace (obtained using 1 000 traces) where the operations of the first round of the underlying AES encryption are marked. The following parts of this section deal with different schemes and methods we used to analyze the vulnerability of the implementation.

Algorithm 1: AES-256 used for the DPA contest V4 [43].

Input : Plaintext X, seen as 16 bytes X_i, $i \in [\![0, 15]\!]$,
Key schedule, 15 128-bit constants $\mathsf{RoundKey}_r$, $r \in [\![0, 14]\!]$
Output: Ciphertext X, seen as 16 bytes X_i, $i \in [\![0, 15]\!]$

Draw a random offset, uniformly in $[\![0, 15]\!]$
$X = X \oplus \mathsf{Mask}_{\mathsf{offset}}$ /* Plaintext blinding */

for $r \in [\![0, 13]\!]$ **do**
  $X = X \oplus \mathsf{RoundKey}_r$ /* AddRoundKey */
  for $i \in [\![0, 15]\!]$ **do**
    $X_i = \mathsf{MaskedSubBytes}_{\mathsf{offset}+i+r}(X_i)$
  end
  $X = \mathsf{ShiftRows}(X)$
  if $r \neq 13$ **then**
    $X = \mathsf{MixColumns}(X)$
    $X = X \oplus \mathsf{MaskCompensation}_{\mathsf{offset}+1+r}$
  end
end

$X = X \oplus \mathsf{RoundKey}_{14}$ /* Last AddRoundKey */
$X = X \oplus \mathsf{MaskCompensationLastRound}_{\mathsf{offset}+14}$ /* Ciphertext demasking */

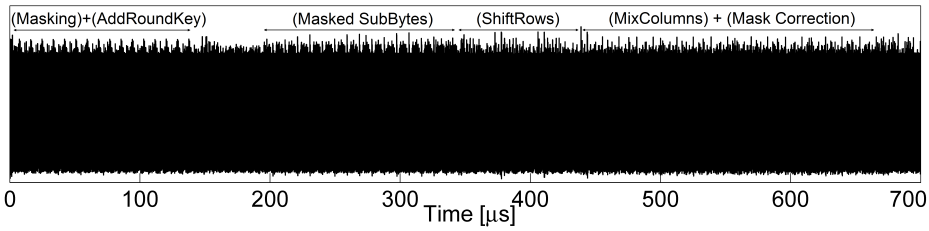


Fig. 1. A mean trace covering the first round of the AES encryption, using 1 000 traces

Examining the First-Order Leakage Back to the original correlation-collision attack [29], which is shortly restated later, the authors proposed a *variance test approach* which can identify the time instances when a first-order leakage is exhibited by the traces. It is worth to mention that relatively-similar approaches were previously introduced in [1,42] as *inter cluster separation* and *variance test*. In order to follow this approach we first need to estimate the mean of the traces classified by the plaintext bytes. To express it formally let us denote the number of traces by N, the plaintexts by $p^0, \ldots, p^{N-1}$, the plaintext bytes by $p^{n\in\{0,\ldots,N-1\}}_{j\in\{0,\ldots,15\}}$, and the traces by $t^0, \ldots, t^{N-1}$. We also express the corresponding random variables as P, $P_{j\in\{0,\ldots,15\}}$, and T. We now estimate the mean traces,

denoted by $m_{j=\{0,\ldots,15\}}^{i=\{0,\ldots,255\}}$, as follows:

$$m_j^i = \mathbb{E}(T|P_j = i) \ .$$

According to [29] a variance trace over the mean traces, e.g., $v_{j\in\{0,\ldots,15\}} = \mathrm{Var}(m_j^i; \forall\, i)$ should indicate the time samples in which the mean traces depend on the plaintext byte, i.e., j. For example, Fig. 2 shows two variance traces v_0 and v_2 obtained using 100 000 traces. Note that according to the realization of the scheme expressed in Section 3.1, the corresponding plaintext bytes of these two variance traces, i.e., 0 and 2, are processed consequently during the SubBytes operation. As clearly shown by the graphics, there is an unambiguous dependency between the mean traces $m_0^{i\in\{0,\ldots,255\}}$ and the value of the first plaintext byte when the relevant S-box is computed. Therefore, due to the initial AddRoundKey and the AES S-box as a bijection, the same dependency holds for the S-box input as well as its output. As a result, back to the definition of a first-order leakage illustrated in Section 1 we conclude that there is a first-order leakage available in the traces. In the next parts of this section we show how to extract this leakage thereby recovering the secrets. We should note that the big peaks shown by Fig. 2 before 50 μs are related to the initial masking of plaintext bytes before the key addition.

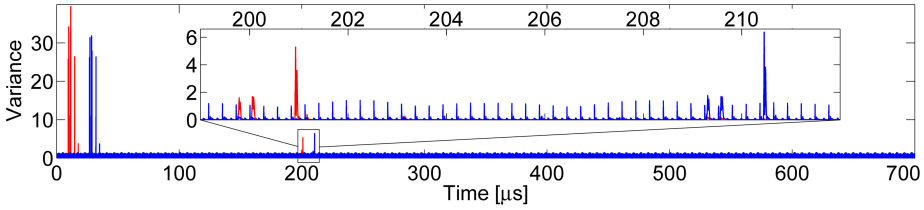


Fig. 2. Two variance traces v_0 and v_2, using 100 000 traces, i.e., around 390 traces per mean trace

Correlation-Collision Attack In order to perform a correlation-collision attack which aims at recovering the linear difference between the targeted key bytes (see AES linear collision attack [7]) the mean traces, e.g., m_0^i and m_2^i, should be first aligned based on the time instances of leaking parts discovered by the variance check approach restated above. Suppose that $m_2'^i$ indicate the mean traces m_2^i which are aligned to the mean traces m_0^i, i.e., by shifting each mean trace m_2^i 9.524 μs (4762 sample points) to the left (see Fig. 2). For a specific key difference guess $\Delta k = k_0 \oplus k_2$, computing the correlation between m_0^i and $m_2'^{i\oplus\Delta k}$ (series of 256 values indexed by i) at each sample point individually leads to a correlation trace $c^{\Delta k}$. Repeating the same scenario for all possible Δk guesses we obtain 256 correlation traces which are shown by Fig. 3. The correct

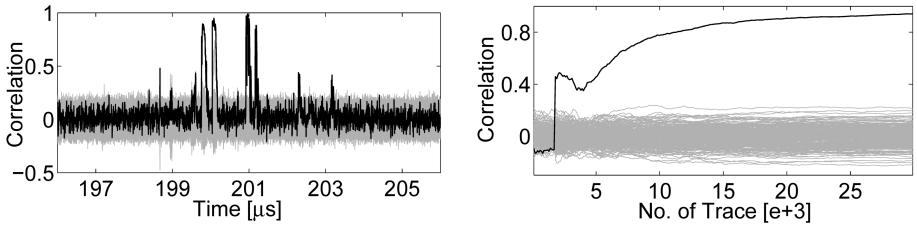
Δk can be clearly distinguished from the other candidates. We repeated this scheme targeting different key bytes, and the difference between all key bytes can be recovered similarly. Moreover, the number of required traces for a successful Δk recovery, reported as 2500 traces by Fig. 3(b), is approximately the same for other key bytes. The achieved correlation is almost reaching one (its maximal value), which is consistent with the reuse of exactly the same code for the evaluation of all the sixteen S-boxes.

We now seek for a faster attack, namely a CPA with a relevant leakage model.

CPA by Bit Model According to the property of the underlying masking scheme and the specific way the mask list is selected [31] (also restated in Section 2), there should not exist any first-order leakage. However, the results shown above somehow contradict with the security proofs. Therefore, we tried to *pinpoint* the leakage source by performing CPA attacks with different hypothetical power models. The straightforward models like the Hamming weight (HW) of the S-box input or its output failed to recover any secret. The same holds for all bit-wise models, e.g., the most significant bit (MSB) of the S-box output. However, our analysis showed a clear dependency between the traces and bit-wise Hamming distance (HD) of the S-box input and output. In other words, when the power model is selected as

$$hp(x, k, b) = (x \oplus k \oplus \mathsf{SubBytes}\,(x \oplus k)) \;\&\; 2^b \;, \tag{1}$$

where x denotes the plaintext byte value, k the key byte candidate, and $b \in \{0, \ldots, 7\}$ the bit position within the byte, the CPA attack is able to recover the correct key candidate. Figure 4 shows the CPA attack results for all 8 bit-wise models targeting the first key byte. As shown by the graphics, the attacks are successful not for all the selected models, and the polarity of the relation between the model and the traces differs from a model to another. For example, the polarity of correlation value for the correct key candidate related to the bits 0 and 3 is the inverse of that of the bits 6 and 7. We should mention that the shape of the graphics and the attack results look similar when targeting other



(a) (b)

Fig. 3. Result of a correlation collision attack targeting the difference between the first and the third key bytes $\Delta k = k_0 \oplus k_2$, (a) using 100 000 traces, (b) at time instance 200.968 μs over number of traces

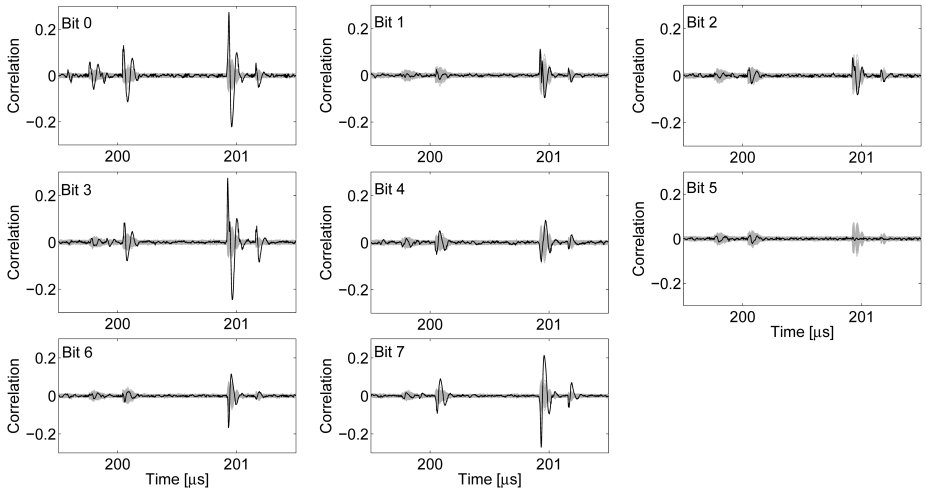


Fig. 4. The CPA attack results, bit-wise HD model of S-box input:output, using 100 000 traces

key bytes. Another issue is related to the low number of required traces, i.e., around 500, to successfully mount the attack.

Leakage Source In order to find the reason behind such leakage we carefully followed the operations performed during the SubBytes operation. As illustrated before the i-th masked S-box, which gets the input masked by M_i, issues the S-box output masked by $M_{i+1 \bmod 16}$. It means,

$$\mathsf{MaskedSubBytes}_i(x') = \mathsf{SubBytes}(x' \oplus M_i) \oplus M_{i+1 \bmod 16},$$

where x' denotes the masked input as $x \oplus M_i$. Since during the SubBytes operation the cipher state is replaced by its substituted one using the S-box, the XOR of the S-box input and output usually influences the power consumption. Following the given formula above, the XOR of a masked S-box input and output yields to

$$\begin{aligned} x' \oplus \mathsf{SubBytes}(x' \oplus M_i) \oplus M_{i+1 \bmod 16} = \\ x \oplus M_i \oplus \mathsf{SubBytes}(x) \oplus M_{i+1 \bmod 16} = \\ x \oplus \mathsf{SubBytes}(x) \oplus M_i \oplus M_{i+1 \bmod 16}. \end{aligned} \tag{2}$$

It means that the XOR between the S-box input and output (for instance if these two values are consecutively saved in a register) is masked by $M'_i = M_i \oplus M_{i+1 \bmod 16}$. Considering the used mask table (see Section 2), M'_i is amongst the list below:

$$\begin{aligned} M'_{i \in [\![0,15]\!]} = \{ &\texttt{0x0f, 0x39, 0x0f, 0x6a, 0x0f, 0x39, 0x0f, 0xff,} \\ &\texttt{0x0f, 0x39, 0x0f, 0x6a, 0x0f, 0x39, 0x0f, 0xff}\} \ . \end{aligned}$$

Table 1. Probabilities of $M'^{(b)}$ being equal to one

$M'^{(b)}$	$M'^{(0)}$	$M'^{(1)}$	$M'^{(2)}$	$M'^{(3)}$	$M'^{(4)}$	$M'^{(5)}$	$M'^{(6)}$	$M'^{(7)}$
$\mathbb{P}(M'^{(b)} = 1)$	0.875	0.750	0.625	1.000	0.375	0.500	0.250	0.125

First, this mask list does belong to the codewords ($[8, 4, 4]$ code) defined in Section 2.4, since they are obtained by the composition with a XOR (the internal law) of pairs of codewords. Second, the distribution of the list does not seem to be a suitable mask list as it consists of 8 times `0x0f`, 4 times `0x39`, 2 times `0x6a`, and 2 times `0xff`. This code M' is not balanced, hence inefficient against first-order attacks. Therefore, the leakage observed by the correlation-collision attack as well as the CPA with bit-wise HD model is due to this fact that the XOR of the S-box input and output is not suitably masked.

CPA by Optimal Model In order to understand the results in Fig. 4, which show very distinct correlation coefficients for each bit, and to identify an optimal model for CPA, we further investigate in the code M'. As in Eq. (1), let $z = x \oplus k \oplus \mathsf{SubBytes}(x \oplus k)$ and $y = \mathrm{HW}(z \oplus m') + N$, with random mask $m' \in M'$ and additive noise N. It is known [37] that, if the noise N is Gaussian, the optimal model is given by $f_{\mathrm{opt}}(Z) = \mathbb{E}(Y|Z)$. We note that m' is uniformly distributed in M', that is a *code with duplicated codewords* (i.e., it is not a *simple* code as M). By linearity of the Hamming weight, we gain the following: (by $z^{(b)}$ and $m'^{(b)}$ we denote the right most b-th bit of z and m' respectively)

$$\begin{aligned} f_{\mathrm{opt}}(z) &= \textstyle\sum_{b=0}^{7} \mathbb{E}(z^{(b)} \oplus m'^{(b)}) \\ &= \textstyle\sum_{b=0}^{7} z^{(b)} \times \mathbb{P}(m'^{(b)} = 0) + (1 - z^{(b)}) \times \mathbb{P}(m'^{(b)} = 1) \\ &= \underbrace{\textstyle\sum_{b=0}^{7} \mathbb{P}(m'^{(b)} = 1)}_{:=\alpha} + \textstyle\sum_{b=0}^{7} z^{(b)} \times 2\left(\mathbb{P}(m'^{(b)} = 0) - \tfrac{1}{2}\right) \quad . \end{aligned} \tag{3}$$

Further, according to M' we can compute the probabilities for each $m'^{(b)}$ equal to 1 as given in Tab. 1, and of course, $\mathbb{P}(m'^{(b)} = 1) = 1 - \mathbb{P}(m'^{(b)} = 0)$.

We can ignore the constant $\alpha = 4.5$ in Eq. (3), as it is not relevant for CPA. Similarly, we can multiply f_{opt} by a constant (e.g., -4) to make all the coefficients be integers. After these transformations, the optimal model is:

$$f_{\mathrm{opt}}(z) = 3\,z^{(0)} + 2\,z^{(1)} + z^{(2)} + 4\,z^{(3)} - z^{(4)} - 2\,z^{(6)} - 3\,z^{(7)} \quad . \tag{4}$$

Note that, we removed the factor for $z^{(5)}$ as the probabilities for both states are 0.5 and thus bit 5 is perfectly masked.

However, the model for CPA given in Eq. (4) is only valid if the assumption $y = \mathrm{HW}(z \oplus m') + N$ is true. Or in other words, if the power consumption is not composed of a weighted sum of bits with different weights, otherwise the probabilities from Tab. 1 have to be adjusted with the weights of the bits. Thus,

in order to identify the weights, we performed a linear regression [22,40] using the model

$$hp(x, b) = (x \oplus k^* \oplus \mathsf{SubBytes}\,(x \oplus k^*) \oplus m') \;\&\; 2^b \;\;, \tag{5}$$

where the mask m' and the correct key k^* are known. Figure 5(a) shows the weights (β-coefficients) estimated by linear regression for each bit $b = \{0, \ldots, 7\}$. Interestingly, one can clearly identify similar β-coefficients for each bit and thus a clear Hamming weight leakage at the last two leakage moments. Note that, these are the same moments as in Fig. 4. Therefore, the assumption on y is valid and we can directly use the weights as in Eq. (4).

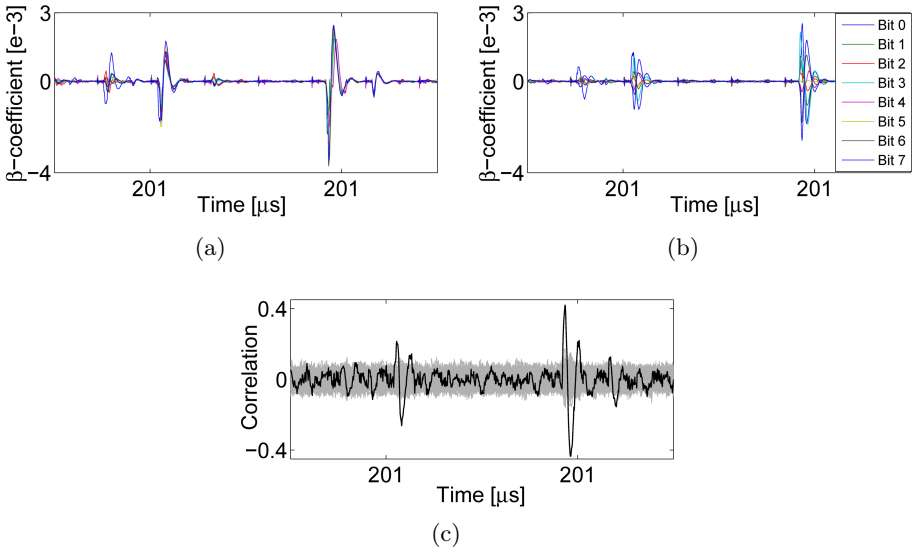


Fig. 5. (a) β-coefficients for bit $b = \{0, \ldots, 7\}$ when the mask and the correct key are known, showing a clear HW leakage at the main leakage moments, (b) β-coefficients when the mask is unknown (c) CPA result using the optimal model Eq. (4) using 1 000 traces

The CPA attack result when using the optimal model (Eq. (4)) is depicted in Fig. 5(c). The graphics show the suitability of the model as the correlation of the correct key candidate is much higher compared to that of Fig. 4 indicating less than 200 traces for a successful attack. Interestingly, the "investment" of 100 000 traces required for the leakage detection (recall Fig. 2) allows a considerable speed-up in the leakage exploitation.

Additionally, the probabilities given in Tab. 1 also explain the results of the bitwise CPA. As explained above bit 5 is perfectly masked, which is also reflected in Fig. 4. Additionally, the greater $|\,\mathbb{P}(M'^{(b)} = 1) - 0.5\,|$ the easier the bit is to

attack, since it is not well masked. For example, the bits with the highest distance are $b = \{0, 3, 7\}$, which also show the highest correlation coefficient in Fig. 4.

Moreover, we perform a linear regression without considering the mask using the model

$$hp(x, b) = \left((x \oplus k^* \oplus \mathsf{SubBytes}\,(x \oplus k^*)) \;\&\; 2^b\right). \tag{6}$$

The β-coefficients are displayed in Fig. 5(b). When looking at the highest leakage moment around 201μs, we can see that coefficient of bit 5 is nearly zeros, thus has no influence as explained before. Furthermore, the other bits follow the same tendency as in Fig. 4 and in Tab. 1. Thus, using the optimal model in Eq. (4) is "equivalent" to the use of a profiled model.

4 Discussion

4.1 Attack and Leakage Orders

The leakage we discovered in the specific implementation of RSM on the ATMega smartcard is based on Hamming distance. Indeed by Hamming distance between a and b we recover the bits of $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$. Therefore, the execution platform itself is realizing the multiplication between several values, that happen to be $x \oplus k \oplus M_i$ on the one hand and $\mathsf{SubBytes}(x \oplus k) \oplus M_{i+1 \mod 16}$ on the other. So, as noted in Eq. (2), a first-order leakage is created by the device itself by artificially multiplying the bits of the S-box input and output. This leakage is subtle in that it is not a trivial unmasking, as if $X \oplus S_0$ would be overwritten by S_0.

Strictly speaking, the *rot is already set in*, meaning that the implementation on the smartcard actually prepares a leakage that can be exploited at first order by an attacker.

4.2 Comparison with Other Attacks on the DPA Contest V4 AES Traces

Ye and Eisenbarth implemented several distribution-based attacks [46]. They exploit the fact that even though the implementation of RSM manages to cancel the moments of order 1, 2 and 3 of the leakage conditioned by a sensitive byte, the moments of order greater than 3 do depend on the sensitive byte. Therefore, the information-theoretic study of the leakage will without doubt allow to put forward biases exploitable in key recovery attacks. In this sense, RSM is not a first-order masking scheme according to the definition that can be found in [11] for instance. A couple of nice and interesting tools, e.g., detecting the collisions due to the complimentary mask lists, are provided in [46] to make use of the leakage distributions. As underlined at [21], the common protection strategy behind *leakage squeezing* [24] and RSM [31], namely the cancellation of leakage moments, indeed conveys an increased security. This explains why the attacks of Ye and Eisenbarth require many more traces (around 10 000), where a first-order attack knowing the mask requires only about 12 traces.

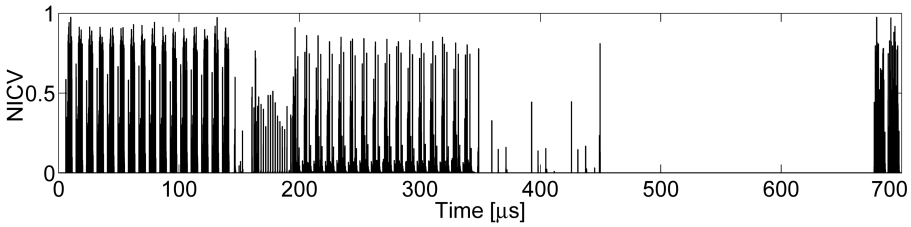


Fig. 6. Normalized inter-class variance for the mask offset of the RSM countermeasure

Lerman et al. have developed a profiling attack that consists in recovering the masks [23]. They used supervised learning to recognize the mask offset, that leaks strongly. This is illustrated in Fig. 6, which shows that the normalized inter-class variance (i.e., NICV = $\mathrm{Var}(\mathbb{E}(T|\mathsf{offset}))/\mathrm{Var}(T)$ [5], also known as the *coefficient of determination*) reaches almost its maximal value '1' at many points in the trace. The idea behind the attack is to make profiles based on the mask value (which has a low entropy), and use these profiles to detect the randomly selected mask during the attack phase and finally run a CPA [8] knowing the mask.

Belgarric et al. prove in [3] that an straightforward second-order correlation attack using the centered product as a combination function [37] needs 300 traces to retrieve the key with probability greater than 80%. Then, the authors assume that the attacker does not know exactly the two leakage points to be combined. Using time-frequency techniques, such as the discrete Hartley transform, the attack remains feasible within about 550 traces even if the investigated window sizes around the leaking samples is of width 2000.

Our attack is particular, in that it requires neither a learning nor a profiling phase. It also does not make use of either the higher-order moments or the leakage distributions. It simply consists in launching a standard attack of lowest possible degree, namely one, with a regular distinguisher (the Pearson correlation coefficient). Our attack, and more precisely the methodology that led to the attack, is *constructive*, in that it allows to point out the leakage cause, which allowed us to fix it (refer to the next Section 4.3).

Summing up, none of the other three attacks referred in this section are specific to RSM, except that of Ye and Eisenbarth (but they require much more traces than a regular second-order attack). Indeed, the attacks of Lerman et al., and Belgarric et al. could as well apply on an FEMS. Since there exists first-order leakage in particular points of the traces, it is not clear whether this leakage was beneficial in the work by Ye and Eisenbarth as well as by Belgarric et al.It is uncertain whether these works are still efficient if the aforementioned first-order leakage is avoided. Our attack is specific to the mask distributions, but what

Table 2. Reordering of the sixteen codewords of $[8, 4, 4]$ linear code so that the Hamming distance between two consecutive codewords is balanced

i	0	1	2	3	4	5	6	7
M_i	0x00	0x0f	0x36	0x39	0x53	0x95	0x5c	0xc9
M_{i+1}	0x0f	0x36	0x39	0x53	0x95	0x5c	0xc9	0xff
$M'_i = M_i \oplus M_{i+1}$	0x0f	0x39	0x0f	0x6a	0xc6	0xc9	0x95	0x36

i	8	9	10	11	12	13	14	15
M_i	0xff	0xc6	0xac	0x9a	0x6a	0xa3	0x65	0xf0
M_{i+1}	0xc6	0xac	0x9a	0x6a	0xa3	0x65	0xf0	0x00
$M'_i = M_i \oplus M_{i+1}$	0x39	0x6a	0x36	0xf0	0xc9	0xc6	0x95	0xf0

it exploits is an unexpected leakage provided by the implementation platform. Concluding, even if all assumptions including the leakage models are hold, the RSM can be insecure *if implemented improperly.*

4.3 Plugging the First-Order Leakage

There are various ways to plug the first-order leakage. We mention hereafter two of them.

- First of all, the masks sequence (S-box input:output relationship) can be tuned, so as to make the HD leakage of Eq. (2) leak-free at first order (which is lower than claimed for RSM, i.e., order 1 versus order 3). We found that there exist several reordering functions $f : \{0, 15\} \rightarrow \{0, 15\}$ of the masks such that $M_i \oplus M_{i+1}$ is balanced. One example of such order is shown in Tab. 2.
- Second, it is easy to identify in the ASM generated by `avr-gcc` the register transfers that are leaking. The leakage source happens to be an instruction `lpm` (Load Program Memory, i.e., read from FLASH) that overwrites its input with its output. We have implemented a secure `lpm` (as an ASM macro) that clears the destination register before it is written to when we access FLASH (where the MaskedSubBytes tables are stored). Still, this implementation (as any implementation of masking) deserves a verification, either with formal methods or with real-world leakage measurements.

5 Conclusions

This paper has highlighted a systematic methodology to detect and then to attack side-channel leakages on cryptographic implementations. The first stage

consists in the *identification*. It can be realized by many tools, such as variance-based tests or NICV [5]. Generally these approaches only detect leakages that involve one or a few bytes of known data (typically plaintext or ciphertext). Indeed, the more bytes, the more traces for the partitioning, and also the more memory for the conditional traces averaging. Another approach, based on pair-wise comparison of traces, has been suggested [20]; however, it requires chosen plaintexts. Second, the attacker will try to turn this leakage into a bias that can yield to a key recovery. The second stage is referred to as the *exploitation*. This step is illustrated in the paper by the intuition that a timely leakage occurring during the S-boxes is likely to have a given expression. A naturally expression (namely the overwriting of a look-up table address by the result) is indeed shown to leak, at first-order (despite the masks that are still there). The efficiency of such an attack is contrasted to second-order attacks [3]. In summary, this paper has shown that a universal verification of the implementation in practice is necessary due diligence, even for provable masking scheme (that are based on hypotheses that must be checked).

Acknowledgements

The authors would like to thank Nicolas Bruneau, from STMicroelectronics Rousset & TELECOM-ParisTech, for the computation of the normalized inter-class variance (NICV) on the offset, and for the research of the masks reordering.

References

1. L. Batina, B. Gierlichs, and K. Lemke-Rust. Differential Cluster Analysis. In *CHES 2009*, volume 5747 of *LNCS*, pages 112–127. Springer, 2009.
2. 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.
3. P. Belgarric, S. Bhasin, N. Bruneau, J.-L. Danger, N. Debande, S. Guilley, A. Heuser, Z. Najm, and O. Rioul. Time-Frequency Analysis for Second-Order Attacks. In *CARDIS 2013*, volume ??? of *LNCS*. Springer, 2013.
4. S. Bhasin, C. Carlet, and S. Guilley. Theory of masking with codewords in hardware: low-weight dth-order correlation-immune Boolean functions. Cryptology ePrint Archive, Report 2013/303, 2013. http://eprint.iacr.org/2013/303/.
5. S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm. NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. In *International Symposium on Electromagnetic Compatibility" (EMC '14 / Tokyo)*. IEEE, May 12-16 2014. Session OS09: EM Information Leakage. Hitotsubashi Hall (National Center of Sciences), Chiyoda, Tokyo, Japan.
6. A. Bogdanov. Improved Side-Channel Collision Attacks on AES. In *SAC 2007*, volume 4876 of *LNCS*, pages 84–95. Springer, 2007.
7. A. Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In *CHES 2008*, volume 5154 of *LNCS*, pages 30–44. Springer, 2008.
8. É. 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.

9. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
10. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Improved Collision-Correlation Power Analysis on First Order Protected AES. In *CHES 2011*, volume 6917 of *LNCS*, pages 49–62. Springer, 2011.
11. J.-S. Coron. Higher Order Masking of Look-up Tables. Cryptology ePrint Archive, Report 2013/700, 2013. http://eprint.iacr.org/.
12. J.-S. Coron and L. Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In *CHES 2000*, volume 1965 of *LNCS*, pages 231–237. Springer, 2000.
13. J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 28–44. Springer, 2007.
14. N. Courtois and L. Goubin. An Algebraic Masking Method to Protect AES Against Power Attacks. In *ICISC 2005*, volume 3935 of *LNCS*, pages 199–209. Springer, 2005.
15. G. Dabosville, J. Doget, and E. Prouff. A New Second-Order Side Channel Attack Based on Linear Regression. *IEEE Trans. Computers*, 62(8):1629–1640, 2013.
16. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In *CARDIS 1998*, volume 1820 of *LNCS*, pages 167–182. Springer, 2000.
17. L. Genelle, E. Prouff, and M. Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In *CHES 2011*, volume 6917 of *LNCS*, pages 240–255. Springer, 2011.
18. B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede. Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis. In *CT-RSA 2010*, volume 5985 of *LNCS*, pages 221–234. Springer, 2010.
19. J. D. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In *CHES 2002*, volume 2523 of *LNCS*, pages 198–212. Springer, 2002.
20. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side-channel resistance validation, September 2011. NIST Non-Invasive Attack Testing Workshop, http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.
21. V. Grosso, F.-X. Standaert, and E. Prouff. Leakage Squeezing, Revisited. In *CARDIS 2013*, volume ??? of *LNCS*. Springer, 2013.
22. O. Kardaun. *Classical Methods of Statistics*. Springer, 2005.
23. L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS 2013*, volume ??? of *LNCS*. Springer, 2013.
24. H. Maghrebi, S. Guilley, and J.-L. Danger. Leakage Squeezing Countermeasure Against High-Order Attacks. In *WISTP*, volume 6633 of *LNCS*, pages 208–223. Springer, June 1-3 2011. Heraklion, Greece. DOI: 10.1007/978-3-642-21040-2_14.
25. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
26. 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.
27. T. S. Messerges. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*. PhD thesis, University of Illinois at Chicago, USA, 2000. 468 pages.

28. T. S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.
29. 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.
30. M. Nassar, S. Guilley, and J.-L. Danger. Formal Analysis of the Entropy / Security Trade-off in First-Order Masking Countermeasures against Side-Channel Attacks. In *INDOCRYPT 2011*, volume 7107 of *LNCS*, pages 22–39. Springer, 2011.
31. M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In *DATE 2012*, pages 1173–1178. IEEE, 2012.
32. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.
33. NIST/ITL/CSD. Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
34. J. Pan, J. I. den Hartog, and J. Lu. You Cannot Hide behind the Mask: Power Analysis on a Provably Secure *S*-Box Implementation. In *WISA 2009*, volume 5932 of *LNCS*, pages 178–192. Springer, 2009.
35. E. Prouff, C. Giraud, and S. Aumônier. Provably Secure S-Box Implementation Based on Fourier Transform. In *CHES 2006*, volume 4249 of *LNCS*, pages 216–230. Springer, 2006.
36. E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In *WISA 2007*, volume 4867 of *LNCS*, pages 227–244. Springer, 2007.
37. E. Prouff, M. Rivain, and R. Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
38. 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.
39. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
40. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer, 2005.
41. K. Schramm and C. Paar. Higher Order Masking of the AES. In *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.
42. F.-X. Standaert, B. Gierlichs, and I. Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers. In *ICISC 2008*, volume 5461 of *LNCS*, pages 253–267. Springer, 2008.
43. TELECOM ParisTech SEN research group. DPA Contest (4^{th} edition), 2013-2014. http://www.DPAcontest.org/v4/.
44. M. Tunstall, C. Whitnall, and E. Oswald. Masking Tables - An Underestimated Security Risk. In *FSE 2013*, volume 8424 of *LNCS*. Springer, 2014. http://eprint.iacr.org/2013/735.
45. J. Waddle and D. Wagner. Towards Efficient Second-Order Power Analysis. In *CHES 2004*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004.
46. X. Ye and T. Eisenbarth. On the Vulnerability of Low Entropy Masking Schemes. In *CARDIS 2013*, volume ??? of *LNCS*. Springer, 2013.