Study of a Novel Software Constant Weight Implementation

Victor Servant[1], Nicolas Debande[2⋆], Houssem Maghrebi[1], Julien Bringer[1]

[1] SAFRAN Morpho,
18, Chaussée Jules César, 95520 Osny, France.
firstname.lastname@morpho.com
[2] SERMA Technologies (ITSEF),
3, avenue Gustave Eiffel, 33608 PESSAC, France.
n.debande@serma.com

Abstract. While in the early 2000's lots of research was focused on Differential Power Analysis of first and second-order, it seems the recent trend is of even higher-order. As this order grows, countermeasures such as masking need to be designed in a more generic way. In this paper, we introduce a new constant weight implementation of the AES extending the idea of the software dual-rail countermeasure proposed by Hoogvorst *et al.* at COSADE 2011. Notably, we illustrate its practicality on 16-bit microcontroller in terms of speed and complexity. This countermeasure applies to all devices that leak a function of the Hamming weight of the internal variables. Under this assumption, our constant weight implementation is theoretically inherently resistant to side-channel attacks of any order. A security evaluation is conducted to analyze its resistance when the leakage slightly deviates from the Hamming weight assumption. It reveals that the countermeasure remains as good as several well-known masking countermeasures. Moreover, the proposed countermeasure offers the possibility to detect some classes of faults.

Keywords: Constant weight, information theoretic analysis, side-channel analysis, AES, software implementation.

1 Introduction

Since the introduction of Differential Power Analysis (DPA) by Kocher [12], Side-Channel Analyses (SCA) have become important issues for the security of cryptographic devices. During the two last decades, a lot of efforts have been dedicated towards the research about SCA and the development of corresponding countermeasures.

A very common countermeasure to protect implementations of block ciphers against SCA is to randomize the sensitive variables by *masking* techniques. The core principle of masking is to ensure that every sensitive variable is randomly

⋆ Work done when the author was at SAFRAN Morpho.

split into at least two shares so that the knowledge of a strict sub-part of the shares does not give information on the shared variable itself. Masking can be characterized by the number of random masks used per sensitive variable. So, it is possible to give a general definition for a d^{th}-order masking scheme: every sensitive variable Z is randomly split into $d+1$ shares $M_0, \cdots, M_d$ in such a way that the relation $M_0 \perp \cdots \perp M_d = Z$ is satisfied for a group operation $\perp$ (*e.g.* the XOR operation in Boolean masking) and no tuple of strictly less than $d+1$ shares depends on Z. Obviously, a d^{th}-order masking can be theoretically defeated by a $(d+1)^{\text{th}}$-order SCA attack that jointly involves all the $d+1$ shares.

In the literature, several provably secure higher-order masking schemes have been proposed, see for instance [3],[23] and [5]. But, due to their large penalty factors (complexity and speed), these countermeasures are unpractical for an everyday use of a smartcard.

In this paper, we perform an in-depth analysis of an alternative to masking countermeasure which consists in coding the data with a fixed *Hamming weight* value, and perform all operations following this fashion. It is often assumed that a device leaks information based on the Hamming weight of the data being operated on, or the Hamming distance between the processed data and its initial state. This assumption is quite realistic and many security analyses in the literature have been conducted following this model [2],[17]. This paper introduces a new variety of *constant weight* codes which can be used to secure software implementations of block ciphers. Typically, we show that assuming a Hamming weight leakage function (or even some small variations of it), it is possible to prevent side-channel attacks.

The rest of the paper is structured as follows. We first recall the two published constant weight (dual-rail) implementations of a block-cipher in software and look into their advantages and drawbacks in Sec. 2. Then, we describe a new solution for a constant weight implementation in Sec. 3, and apply it to the AES in Sec. 4. Finally, we conduct an information theoretic analysis in Sec. 5, and a security analysis in Sec. 6 to evaluate our proposed scheme. The conclusion and some perspectives are in Sec. 7.

2 Previous works

Hoogvorst *et al.* [10] presented a *dual-rail* implementation of PRESENT [1] in 2011. The paper aimed at proving that one could rather easily protect a block cipher in software using constant weight coding style rather than using masking. The idea was straightforwardly taken from existing dual-rail hardware, and consists in encoding one bit *s.t.* $0 \rightarrow 01$ and $1 \rightarrow 10$ (or the inverse).

The adaptation of this solution to software implementation required computing tables performing all basic operations such as XOR, AND, *etc.* In the end, the execution is 9 times slower than their original unsecured implementation. The memory cost is minimal: only very little non-volatile memory is required to store the tables (256 for the Sbox and 16 bytes for the XOR operation) and an unchanged RAM cost. Given the theoretical protection offered by such an imple-

mentation, it seems a very attractive choice cost-wise. Note that the complexity to achieve such a protection was minimal thanks to the very lightweight structure of PRESENT (only two types of operations used) and to the assumption that the expanded keys were already in dual-rail representation. No side-channel analysis was conducted and it is argued that this coding style seems limited to lightweight ciphers, or AES in a different field representation.

The authors in [8] introduce an improvement on the previous idea which drastically simplifies the XOR operation. Moreover, a side-channel and performance analysis of PRESENT and Lightweight Block cipher under this form are presented. No vulnerability appears when targeting the S-box output. The overhead in execution time is almost negligible.

However, the trick used to accelerate a XOR operation induces a leakage. The authors noticed that for any pair of variables (A, B), we have $C(A) \oplus C(B) \oplus 01 = C(A \oplus B)$, where C denotes the chosen dual-rail code. Performing a constant weight XOR does not require an access to a precomputed table this way. They argue that performed in some precise order, these operations do not leak a potential secret value. This works if one assumes there is only a single secret value XORed with a non-secret. Unfortunately, this assumption cannot be made for the second round of PRESENT and for the AES, as the XORs performed during the first MixColumns operation contain information on the secret key in both operands, making a side-channel attack possible.

In [22], Rausy *et al.* present a balancing strategy based on the execution of binary operators only. This balancing protection uses *dual-rail* with two bits in the registers, selected to be as similar as possible in terms of leakage, and S-Boxes are computed using a bit-slice representation.

All the aforementioned works tried to enforce the dual-rail representation. In this paper, we turn our attention to other classes of constant weight strategy.

3 A constant weight AES

3.1 The AES algorithm

The AES [15], formerly known as Rijndael has been the international standard for symmetric ciphers and much research has focused on securing it against side-channel attacks since its adoption in 2000. It can be described as having four layers: a Substitution (*SubBytes*), a permutation (*ShiftRows*), a mixing phase (*MixColumns*) and a Key Addition (*AddRoundKey*).
The *SubBytes* phase is a nonlinear transformation in $GF(2^8)$ and is often implemented as a table lookup in software. *ShiftRows* is simply a sequence of byte swaps. *MixColumns* is a matrix multiplication of all 16 bytes of the AES state with a constant matrix, it can be implemented as several bitwise XORs and field multiplications. Those multiplications are based on an operation called $Xtimes$, which is the multiplication of a polynomial (represented as a byte) by X over $GF(2^8)$. This procedure is a simple bit test and a shift, plus a conditional XOR with a constant. It could also be implemented as a table look-up to avoid timing

attacks. The last operation, *AddRoundKey*, is a simple XOR between the state values and a round key.

3.2 Constant weight codes

Constant weight codes have a simple definition: it is any code that has all its words of a given weight, say ω. In the following, we denote (x,y)-code the set of values of weight x over y bits, which contains $\binom{y}{x}$ elements. The dual-rail representation is a specific case of these codes, but it is not the only option one should consider in a software setting. It is adapted to the hardware environment as one has to deal with pair-wise balancing of wires. In software, one could simply use the code with the smallest cardinal available to encode the input set of data. A 4-bit data set contains 16 elements. The $(3, 6)$-code presented in Tab. 1 (the set of 6-bit words of Hamming Weight 3) contains 20 elements, and is therefore large enough to encode the previous 4-bit set. Encoding (non-linearly) in this way could simply be a random assignment. For the rest of this paper, we will refer to the $(3, 6)$-code simply by $\mathcal{C}$.

Table 1. $(3, 6)$-code

$0 \rightarrow$ 000111	$4 \rightarrow$ 010011	$8 \rightarrow$ 011010	$12 \rightarrow$ 100110
$1 \rightarrow$ 001011	$5 \rightarrow$ 010101	$9 \rightarrow$ 011100	$13 \rightarrow$ 101001
$2 \rightarrow$ 001101	$6 \rightarrow$ 010110	$10 \rightarrow$ 100011	$14 \rightarrow$ 101010
$3 \rightarrow$ 001110	$7 \rightarrow$ 011001	$11 \rightarrow$ 100101	$15 \rightarrow$ 101100

3.3 Encoded operations

Let us denote by $\mathcal{C}(A)$ (respectively $\mathcal{C}(B)$) the encoded form of the variable A (respectively B) in a constant weight representation. Then, the operation $A \perp B$ (where $\perp$ is any operation like XOR, AND *etc.*) can be performed in a non-leaking manner using a precomputed table T such that: $T[(\mathcal{C}(A) \ll n) \,||\, \mathcal{C}(B)] = \mathcal{C}(A \perp B)$, where n is either the size of the codewords (*e.g.* $n = 6$ for the $(3, 6)$-code) or the size of a processor word (*i.e.* $n = 8$) That is, if we prepare an index in this table by appending one encoded value to the other and then fetch the result from T, we get the encoded result of $A \perp B$.

For AES, we have to encode 8-bit values. Straightforwardly done, it would take up to 16 bits per state byte in dual-rail. The table for the S-box precomputed to fit this code would span 128 Kbytes of data, which is not a reasonable option for a conventional smartcard. Instead, in [10] authors propose to use the $GF(2^4)$ representation of AES. The S-box is then performed as a sequence of inverses

and multiplications in that same field. This variant is expected to perform slowly due to these operations, see [16] for example. We aim to provide an alternative that performs fast and occupies an acceptable memory space.

For AES block cipher, the smallest code that can encode all of the 256 possible elements of the state is the $(5, 11)$-code (462 elements). The table for performing the S-box would be indexed by 11 bits, thereby spanning 2048 elements of 11 bits each, which would amount to 4 KBytes in a realistic implementation. This is acceptable, but the problem arises from the XOR operation. In dual-rail, it could be done 2 bits by 2 bits, but with the $(5, 11)$-code it is not possible anymore, as this encoding is non-linear. To perform a XOR, a 22 bits index is needed under this form. Of course, this exceeds by far the capacity of any smartcard, so this code is a bad choice. Instead of coding a whole 8-bit variable W into a constant weight word, we split it into two 4 bits words (a high word HB and a low word LB) and encode each of them separately, but using the same $\mathcal{C}$:

$$W = \underbrace{0011}_{\text{HB}}\ \underbrace{1011}_{\text{LB}}, \quad \mathcal{C}(W) = \underbrace{001110}_{\mathcal{C}(\text{HB})}\ \underbrace{100101}_{\mathcal{C}(\text{LB})}$$

This way, linear operations can be performed on the two halves separately at a much lower memory cost. The table for the S-box is now indexed by $6+6 = 12$ bits, which is 4096 elements, and it is the same cost for the XOR. The operations cannot be made at once in this case though. As the Arithmetic Logic Unit (ALU) can only retrieve a byte from memory, we need two accesses to obtain both the HB and the LB of the S-box result. We end up with three tables of 4 KBytes each : one for the S-box's high bytes, one for the low bytes, and one for the XOR. The instruction sequence performing a XOR between two $(3, 6)$-codewords A and B, equal respectively to $(aaaaaa)$ and $(bbbbbb)$ in binary, is shown in Listing 1.1. Displayed on the right is the state of the 16-bit registers used. We stress the fact that line 5 is here to prevent any leakage in the Hamming distance model.

```
1  mov     ax, #0              // ax = 00000000 00000000
2  mov     Rh, A               // ax = 00aaaaaa 00000000
3  mov     Rl, B               // ax = 00aaaaaa 00bbbbbb
4  xor     ax, &table          // ax = ddaaaaaa ddbbbbbb
5  mov     bx, #0              // bx = 00000000 00000000
6  mov     bx, [ ax ]          // bx = 00000000 00cccccc = C(A xor B)
```

Listing 1.1. Table accesses in constant weight (x86 assembly style)

All performed operations meet the constant Hamming weight specifications. The constraints on the address format (2 bits set in the middle (dd)) is easily treated by modern compilers, which usually allow declaring tables at a specific address (*e.g.* the IAR compiler with the @ keyword).

4 Encoding the AES

In this section, we show how the various operations of the AES could be implemented in a constant weight fashion. There are mainly three different types of operations:

(i) Non-linear transformations of one word, *i.e.* SubBytes;
(ii) Two-operand operations, *i.e.* XOR;
(iii) Linear transformations of one word, *i.e.* Xtimes.

In the sequel, we denote by A_h (respectively A_l) the most significant (respectively the least significant) 4 bits of a byte A ($A = A_h \,||\, A_l$).

Computation of type (i). The SubBytes operation will be performed in two accesses : one for the MSB of the result which will be put in register R_h, and another for the LSB which will be stored in register R_l such that:

$$R_h \leftarrow high\_subbytes[\ (\mathcal{C}(A_h) \ll 8) \,||\, \mathcal{C}(A_l)\ ] = \mathcal{C}(SubBytes(A_h))\ ,$$
$$R_l \leftarrow low\_subbytes[\ (\mathcal{C}(A_h) \ll 8) \,||\, \mathcal{C}(A_l)\ ] = \mathcal{C}(SubBytes(A_l))\ .$$

Computation of type (ii). It is a similar case for the XOR operation, but it needs two operands A and B:

$$R_h \leftarrow xor\_table[\ (\mathcal{C}(A_h) \ll 8) \,||\, \mathcal{C}(B_h)\ ] = \mathcal{C}(A_h \oplus B_h)\ ,$$
$$R_l \leftarrow xor\_table[\ (\mathcal{C}(A_l) \ll 8) \,||\, \mathcal{C}(B_l)\ ] = \mathcal{C}(A_l \oplus B_l)\ .$$

Computation of type (iii). At first, one could implement Xtimes just like the S-box as a one-operand full-length table access. This would add 8 more KBytes to the implementation and disregard the linearity of the operation. Instead, one can write the matrix M of Xtimes as:

$$M = \underbrace{\begin{pmatrix} 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&1\\ 0&0&1&0&0&0&0&1\\ 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1 \end{pmatrix}}_{M_h}\underbrace{\phantom{\begin{pmatrix} 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0\\ 0&0&0&0 \end{pmatrix}}}_{M_l}$$

where M_h and M_l are two (8×4) matrices such that $M = [M_h||M_l]$. Then, this linear operation consists in a XOR of two products of an (8×4) matrix by a 4-bit vector:

$$M \cdot A = (M_h \cdot {A_h}^T) \oplus (M_l \cdot {A_l}^T)\ .$$

The necessary tables for M_h and M_l only use 256 bytes of non-volatile memory in total, which is almost negligible compared to the S-box .

4.1 Implementation performance and comparison

The whole implementation had to be done in assembly and using several macros. The code could be smaller as loops had to be fully unrolled - our macros could not easily allow use of variable indexes. The execution time is also reduced for the same reason. The Key Expansion phase was on-the-fly and in constant weight coding as well.
Encoded bytes were always written on registers previously set to 0, thereby preventing any Hamming distance leakage of the form $\mathcal{C}(A) \oplus \mathcal{C}(B)$, which is not constant weight. We compared our $(3, 6)$-code version to a C version of the AES on the same platform. For a fair comparison, it was checked that the compiler did optimize the unprotected version as much as possible. The results are enclosed in Tab. 2.

Table 2. Implementation results

Version	Speed (Cycles)	Code size (Ko)
AES unprotected	3.490	2,8
AES using $(3, 6)$-code	14.625	24,0

From Tab. 2, one can conclude that the protected version of the AES using $(3, 6)$-code is about $4, 2$ times slower, and $8, 5$ times bigger in code size.

In the following, we compare this implementation to existing higher-order masking schemes applied to block ciphers. As the targeted platforms are different we can only evaluate the performance in terms of loss compared to an unprotected version on the same platform, hence the scores in Tab. 3 are given as factors (*e.g.* $\times 2$ means the secured implementation is twice as slow as the unprotected one on the same microcontroller).

From Tab. 3, one can see that the Rivain-Prouff masking scheme of order 3 applied to AES takes 271.000 cycles according to [19], whereas the unprotected AES takes only 2.000 cycles. The performance loss would be $\times 135$ in this case.

4.2 Fault detection capability and memory consumption

Only 256 bytes of each 4 KBytes table will be used by the cipher during normal operation. This seems like a waste of space but actually yields a very interesting feature of this countermeasure which is fault detection.

We filled all unused values of our tables with 0's, a value which is not in the $(3, 6)$-code. If a random byte fault occurs on the state of the AES, then the 0 value will be returned with probability $(4096 - 256)/4096 = 93, 75\%$. This makes a variant of an infective computation method [6], as the 0 will propagate to all future operations within the cipher and the ciphertext will have 0 values in

Table 3. Theoretical resistance of higher-order masking and constant weight schemes against attacks in the Hamming Weight or Distance model.

Method and cipher	Resistance Order	Speed loss	Fault Detection
Higher-order masking schemes			
Masking (AES) [9]	1	× 1,7	.
Rivain-Prouff (AES) [19]	1	×64,0	.
Kim-Hong-Lim (AES) [11]	3	×41,0	.
Genelle-Prouff (AES) [5]	3	×90,0	.
Rivain-Prouff (PICARO) [19]	2	× 6,1	.
Constant weight Schemes			
Dual-rail (PRESENT) [10]	Any	× 9,0	93,75 %
(3,6)-code (AES)	**Any**	**× 4,2**	**93,75%**

place of key-dependent values for the affected bytes. Testing whether a fault was injected or not incurs no overhead (simple zero-test of ciphertext bytes). Also, any one-bit fault can be detected. This fault detection rate provides a strong advantage over all classical masking schemes, which do not inherently provide this detection capability.

Another advantage worth mentioning is that the RAM cost of this constant weight implementation is limited to 64 bytes (instead of 32 for the unprotected variant). Although RAM costs increases with the order of the masking schemes, in our case it is constant for any order.

5 Information theoretic analysis

As argued on the evaluation framework introduced in [24], the robustness of a countermeasure encompasses two dimensions: its amount of leakage irrespective of any attack strategy and its resistance to specific attacks. So, the evaluation of protected implementations should hold in two steps. First, an information theoretic analysis determines the actual information leakage. Second, a security analysis determines the efficiency of various attacks in exploiting this leakage.

Following this evaluation framework, we start with an information theoretic analysis. Under the Hamming weight assumption, it is obvious that the constant weight countermeasure is leakage-free. In fact, the mutual information is null since all manipulated variable have a constant Hamming weight. Therefore we investigate the consequences of a leakage function deviating from the Hamming Weight assumption on our proposed countermeasure. For instance, we assume

that the leakage function is a polynomial of higher degree. Actually, the assumption that all the bits leak identically and without interfering does not hold in real hardware [25]. Also, it has been shown that with specific side channel capturing systems the attacker can distort the measurement. For instance, in [18], the authors show that with a home-made magnetic coil probing the circuit at a crucial location, the rising edges can be forced to dissipate 17% more than the falling edges.

Thus, we study how the the constant weight countermeasure is resilient to imperfections of the leakage model. To do so, we consider the leakage function used in [7], *i.e.* which is a polynomial one of the form:

$$L(Z) = \sum_i a_i \cdot z_i + \sum_{i,j} b_{i,j} \cdot (z_i \cdot z_j) + \sum_{i,j,k} c_{i,j,k} \cdot (z_i \cdot z_j \cdot z_k) \ , \qquad (1)$$

where z_i denotes the i^{th} bit of the sensitive value Z and a_i, $b_{i,j}$ and $c_{i,j,k}$ are some weighting coefficients.

To evaluate the information revealed by the constant weight countermeasure, we compute the *Mutual Information Metric* (MIM) between the sensitive variable and the leakage function under two conditions:

1. **First case:** All bits of a sensitive variable leak identically, but interfere with each other (*i.e.* in Eqn. (1) $\forall i \ \ a_i = a \in \{0,1\}$, $\forall i,j \ \ b_{i,j} = b \in \{0,1\}$, $\forall i,j,k \ \ c_{i,j,k} = c \in \{0,1\}$).
2. **Second case:** The bits of a sensitive variable leak randomly and interfere with each other (*i.e.* in Eqn. (1) $\forall i \ \ a_i \in \{0,1\}$, $\forall i,j \ \ b_{i,j} \in \{0,1\}$, $\forall i,j,k \ \ c_{i,j,k} \in \{0,1\}$).

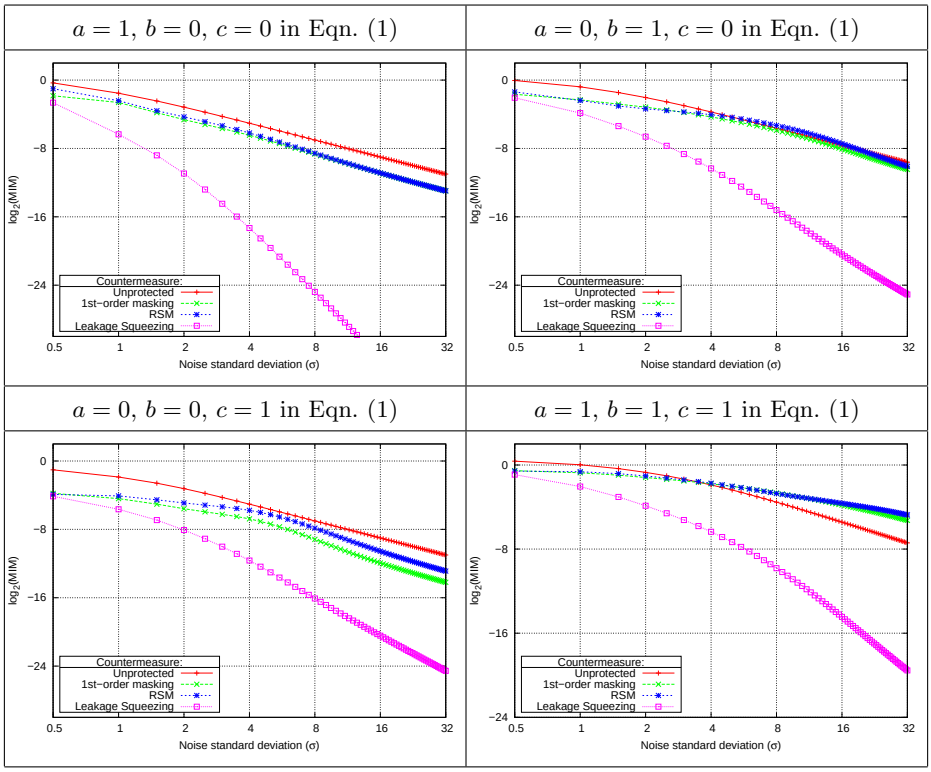
For the sake of comparison, we proceed similarly for several well-known countermeasures. We list hereafter the considered implementations with their corresponding leakage functions:

- Unprotected implementation : $O = L(Z) + N$, where N is a normally distributed noise variable of standard deviation σ (*i.e.* $N \sim \mathcal{N}(0, \sigma^2)$).
- Rotating S-box Masking (RSM) [14]: $O = L(Z \oplus M') + N$, where M' is a low entropy mask chosen within a code.
- Classical first-order Boolean masking[3]: $O = L(Z \oplus M) * L(M) + N$, where M is a full entropy mask.
- Leakage Squeezing[4] [13]: $O = L(Z \oplus M) + L(B(M)) + N$, where B is a bijection chosen within a binary code as well.
- Dual-rail [10] : $O = L(D(Z)) + N$, where D is the dual-rail encoding.
- $(3,6)$-code : $O = L(\mathcal{C}(Z)) + N$.

[3] For this implementation, the leakage corresponds to a bivariate attack, when the product combination is used by the adversary.

[4] This leakage function corresponds to a hardware implementation. To the best of our knowledge, the leakage squeezing countermeasure has never been adapted into a software implementation, therefore we only consider an univariate leakage in our simulation.

Table 4. MIM for polynomial leakage functions with perfect bits transition.



5.1 First case: perfect bits transition

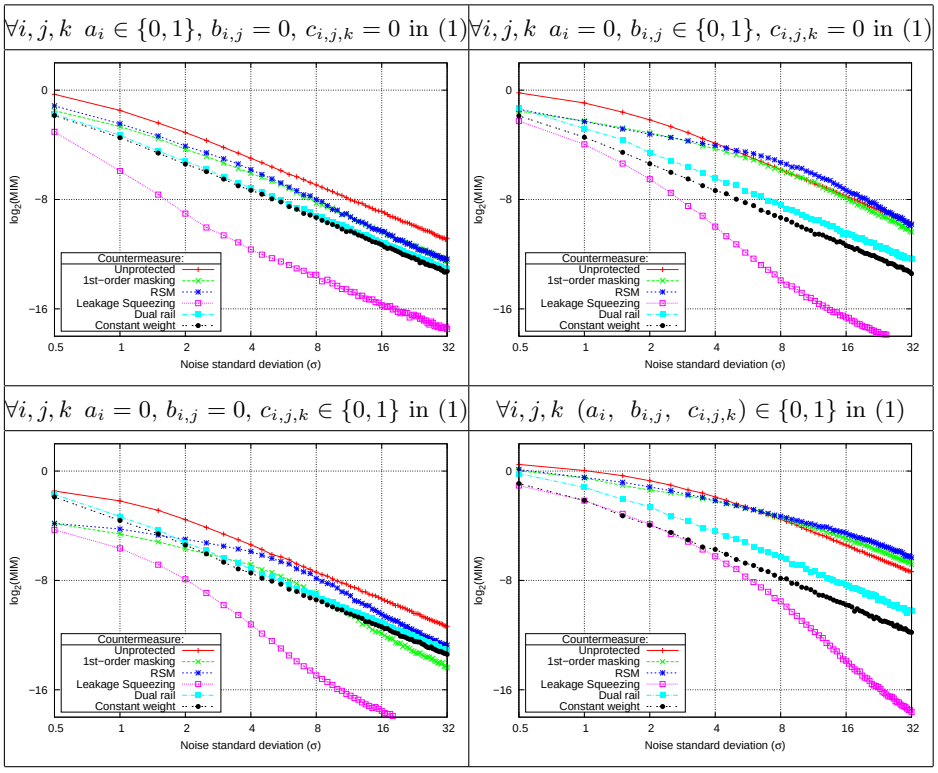
The results are shown in Tab. 4. It is noteworthy that the mutual information for our constant weight countermeasure, as well as for the dual-rail countermeasure, is null whatever σ is. In fact, if all bits leak identically, the leakage function is always constant independently of the values of (a,b,c) and its degree has no influence on the quantity of information leaked. However, the results of our investigation show that for all other countermeasures, the higher the degree of the leakage function, the higher the leaked information. For instance, if the leakage function is a cubic one (*i.e.* $c = 1$), the RSM and the first-order masking lead to a first-order security against bivariate side-channel attacks since the slope of their corresponding MIM curves is equal to 2. Furthermore, these curves are parallel to the one of the unprotected implementation. Concerning the leakage squeezing countermeasure, it ensures a second-order security against univariate side-channel attacks (*i.e.* the slope is equal to 3).

5.2 Second case: random bits transition

In this case, we consider that the bits leak differently. From the results plotted in Tab. 5, the following observations can be emphasized:

- Our proposed countermeasure offers a first-order resistance against univariate side-channel attacks and remains all the same as good as the first-order Boolean masking and the RSM countermeasure. In fact, their corresponding MIM curves have a slope equal to 2.
- When considering high noise values, the quantity of mutual information leaked by the constant weight countermeasure is lower than a first-order masking, for instance. Hence, a first-order attack will succeed, but the adversary will need more traces when dealing with the constant weight countermeasure.

Table 5. MIM for polynomial leakage functions with random bits transition.



6 Security analysis

As a complement to the information theoretic analysis carried out in Sec. 5, we conduct in this section a security analysis to evaluate the resistance of our proposed countermeasure.

6.1 Higher-order side-channel resistance in the Hamming weight model

To prove the resistance of our countermeasure against higher-order SCA attacks in the perfect Hamming model, we have computed the *optimal correlation coefficient* defined in [21] by $f_{\text{opt}}(z) = E[\ (O(Z) - E[O(Z)])^d \mid Z = z\ ]$, where $O(Z)$ denotes the leakage function on the sensitive variable Z and satisfies $O(Z) = HW(\mathcal{C}(Z)) + N$. The noise is denoted by $N \sim \mathcal{N}(0, \sigma^2)$. Then, the optimal correlation coefficient rewrites:

$$\begin{aligned} f_{\text{opt}}(z) &= E[\ (HW(\mathcal{C}(Z)) + N - E[HW(\mathcal{C}(Z)) + N])^d \mid Z = z\ ] \\ &= E[\ (HW(\mathcal{C}(Z)) + N - HW(\mathcal{C}(Z)) - E[N])^d \mid Z = z\ ] \\ &= E[\ N^d\ ]\ . \end{aligned}$$

The last equality is only dependent on the noise, not on the sensitive variable $Z = f(X, K)$, where f denotes any operation using the input variable X and the key K. In that case this means the attack does not work, independently of the order d. Note that switching from Hamming weight leakage protection to Hamming distance protection only requires setting destination registers to 0 before storing the result of a sensitive operation into them. Therefore, this security analysis applies for both leakage models, given a proper implementation.

6.2 Side-channel resistance in the imperfect model

In this section, we evaluate the soundness of the proposed constant weight implementation when the leakage slightly deviates from the rules involved to design this countermeasure. First, we analyse if our implementation shows some vulnerabilities against first-order CPA attack, and then we examine how robust it is against a stochastic online attack [4]. The purpose of this stochastic approach is to deduce the global activity associated to arbitrary chosen events occurring during the encryption (typically a bit-flipping). Although the Hamming weight of each manipulated word remains constant at any time, we expect that the stochastic online approach can exploit differences from a bit register to another, especially if the leakage function deviates from the Hamming weight model as highlighted in the previous information theoretic analysis.

For comparison purpose, we computed the success rate of CPA and stochastic online attack on an unsecured software implementation of AES over 1.000 independent experiments. Concerning the constant weight AES implementation,

we performed these *distinguishers* over 20.000 independent experiments. In our practical attack scenario, we considered the following simulated leakage model:

$$O = \sum_{i=1}^{8} a_i \cdot z_i + N \ , \tag{2}$$

where a_i are some weighting coefficients following a Gaussian law $\mathcal{N}(1, \sigma_e)$, z_i is the i^{th} bit of the sensitive value Z (equals S-box$[X \oplus K]$ for the unprotected AES, and equals $\mathcal{C}$(S-box$[X \oplus K]$) for the $(3, 6)$-code constant weight implementation) and N is an environmental noise *s.t.* $N \sim \mathcal{N}(0, \sigma)$. This model allows us to simulate the leakages by taking into account a slight deviation from the Hamming weight leakage model. The results of these attacks are shown on Fig. 1 for $\sigma_e = 0.1$ and $\sigma = 2$.

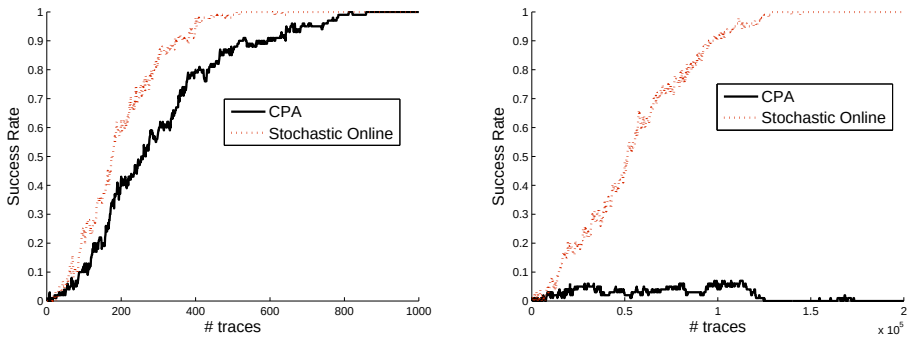


Fig. 1. CPA and stochastic online attacks on unsecured AES implementation (left) and constant weight implementation (right)

From Fig. 1 the following observations can be emphasized:

- As expected, the CPA attack is no longer efficient on the secured implementation even if the leakage model deviates from the Hamming weight assumption.
- Considering the stochastic online attack results, one can see that the unprotected implementation is easily broken. In fact, about 400 traces suffices to achieve a success rate of 100%. As expected and revealed by the information theoretic analysis, the $(3, 6)$-code implementation performs worse when the bit-flipping is random. Indeed, the success rate of the stochastic online attack reaches 100% with 140K traces, although this represents a gain of robustness of about a factor 350.

7 Conclusion

An investigation on whether the AES could be implemented in a constant weight fashion on a 16-bit smartcard was conducted. Instead of using a dual-rail code, we chose an "m out of n" code that enables fast operations at an acceptable memory cost. We have argued that our proposal is a leak-free countermeasure under some realistic assumptions about the leakage model. The solution has been evaluated within an information-theoretic study, proving its security against SCA under the Hamming weight assumption. When the leakage function deviates slightly from this assumption, our solution still achieves good results. On the performance side, it was shown our $(3, 6)$-code AES is faster at execution than most generic higher-order masking schemes, and also comes with some fault detection capability at no cost; a feature which masking schemes lack.

Acknowledgments

This work has been partially funded by the ANR projects E-MATA HARI and SPACES.

References

1. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, September 10-13 2007. Vienna, Austria.
2. É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.
3. J.-S. Coron. Higher Order Masking of Look-Up Tables. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.
4. J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.
5. L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Preneel and Takagi [20], pages 240–255.
6. B. Gierlichs, J.-M. Schmidt, and M. Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. Cryptology ePrint Archive, Report 2012678, 2012. `http//:eprint.iacr.org`.
7. V. Grosso, F.-X. Standaert, and E. Prouff. Low Entropy Masking Schemes, Revisited. In A. Francillon and P. Rohatgi, editors, *CARDIS*, volume 8419 of *LNCS*, pages 33–43. Springer, 2013.
8. Y. Han, Y. Zhou, and J. Liu. Securing lightweight block cipher against power analysis attacks. *Future Wireless Networks and Information Systems*, pages 379–390, 2012.
9. C. Herbst, E. Oswald, and S. Mangard. An AES Smart Card Implementation Resistant to Power Analysis Attacks. In J. Zhou, M. Yung, and F. Bao, editors, *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

10. P. Hoogvorst, G. Duc, and J.-L. Danger. Software implementation of dual-rail representation. *COSADE*, 2011.
11. H. Kim, S. Hong, and J. Lim. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In Preneel and Takagi [20], pages 95–107.
12. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages pp 388–397. Springer, 1999.
13. H. Maghrebi, S. Guilley, C. Carlet, and J.-L. Danger. Optimal First-Order Masking with Linear and Non-Linear Bijections. In *AFRICACRYPT*, LNCS. Springer, July 10-12 2012. Al Akhawayn University in Ifrane, Morocco.
14. M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger. RSM: a Small and Fast Countermeasure for AES, Secure against First- and Second-order Zero-Offset SCAs. In *DATE*, pages 1173–1178. IEEE Computer Society, March 12-16 2012. Dresden, Germany. (TRACK A: "Application Design", TOPIC A5: "Secure Systems").
15. NIST/ITL/CSD. Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
16. E. Oswald and K. Schramm. An efficient masking scheme for aes software implementations. In J.-S. Song, T. Kwon, and M. Yung, editors, *Information Security Applications*, volume 3786 of *Lecture Notes in Computer Science*, pages 292–305. Springer Berlin Heidelberg, 2006.
17. É. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater. Improved Higher-Order Side-Channel Attacks With FPGA Experiments. In *CHES*, volume 3659 of *LNCS*, pages 309–323. Springer-Verlag, 2005. Edinburgh, UK.
18. É. Peeters, F.-X. Standaert, and J.-J. Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, The VLSI Journal, special issue on "*Embedded Cryptographic Hardware*"*, 40:52–60, January 2007. DOI: 10.1016/j.vlsi.2005.12.013.
19. G. Piret, T. Roche, and C. Carlet. PICARO – A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance. In F. Bao, P. Samarati, and J. Zhou, editors, *ACNS*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2012.
20. B. Preneel and T. Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings*, volume 6917 of *LNCS*. Springer, 2011.
21. E. Prouff, M. Rivain, and R. Bévan. Statistical analysis of second order differential power analysis. Cryptology ePrint Archive, Report 2010/646, 2010. `http://eprint.iacr.org/`.
22. P. Rauzy, S. Guilley, and Z. Najm. Formally Proved Security of Assembly Code Against Leakage. *IACR Cryptology ePrint Archive*, 2013:554, 2013. (Also appears at PROOFS 2014, Busan, South Korea).
23. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
24. F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.
25. N. Veyrat-Charvillon and F.-X. Standaert. Mutual Information Analysis: How, When and Why? In *CHES*, volume 5747 of *LNCS*, pages 429–443. Springer, September 6-9 2009. Lausanne, Switzerland.