Very High Order Masking: Efficient Implementation and Security Evaluation

Anthony Journault, François-Xavier Standaert

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium
e-mails: anthony.journault, fstandae@uclouvain.be

Abstract. In this paper, we study the performances and security of recent masking algorithms specialized to parallel implementations in a 32-bit embedded software platform, for the standard AES Rijndael and the bitslice cipher Fantomas. By exploiting the excellent features of these algorithms for bitslice implementations, we first extend the recent speed records of Goudarzi and Rivain (presented at Eurocrypt 2017) and report realistic timings for masked implementations with 32 shares. We then observe that the security level provided by such implementations is uneasy to quantify with current evaluation tools. We therefore propose a new "multi-model" evaluation methodology which takes advantage of different (more or less abstract) security models introduced in the literature. This methodology allows us to both bound the security level of our implementations in a principled manner and to assess the risks of overstated security based on well understood parameters. Concretely, it leads us to conclude that these implementations withstand worst-case adversaries with $> 2^{64}$ measurements under falsifiable assumptions.

1 Introduction

The masking countermeasure is among the most investigated solutions to improve the security of cryptographic implementations against side-channel analysis. Concretely, masking amounts to perform cryptographic operations on secret shared data, say with d shares. Very summarized, it allows amplifying the noise in the physical measurements (hence the security level) exponentially in d, at the cost of quadratic (in d) performance overheads [27, 38]. As discussed in [25], these performance overheads may become a bottleneck for the deployment of secure software implementations, especially as the number of shares increases – which is however needed if high security levels are targeted [15].

In this respect, two recent works from Eurocrypt 2017 tackled the challenge of improving the performances of masked implementations. In the first one, Goudarzi and Rivain leveraged the intuition that bitslice implementations are generally well suited to improve software performances, and described optimizations leading to fast masked implementations of the AES (and PRESENT), beating all state-of-the-art implementations based on polynomial representations [22]. In the second one, Barthe et al. introduced new masking algorithms that are perfectly suited for parallel (bitslice) implementations and analyzed the formal security guarantees that can be expected from them [5].

Building on these two recent works, our contributions are in four parts:

First, since the new masking algorithms of Barthe et al. are natural candidates for bitslice implementations, we analyze their performance on a 32-bit ARM Cortex M4 processor. Our results confirm that they allow competing with the performances of Goudarzi and Rivain with limited optimization efforts.

Second, we put forward the additional performance gains that can be obtained when applying the algorithms of Barthe et al. to bitslice ciphers with limited non-linear gates, such as the LS-design Fantomas from FSE 2014 [23].

Third, and since our implementations can run with very high number of shares (we focus on the case with $d = 32$), we question their security evaluation. For this purpose, we start from the observation that current evaluation methodologies (e.g., based on leakage detection [21, 10, 33, 44, 16] or on launching high order attacks [49, 39, 35]) are not sufficient to gain quantitative insights about the security level of these implementations (and the risks of errors in these evaluations). Hence, we introduce a new "multi-model" methodology allowing to mitigate these limitations. This methodology essentially builds on the fact that by investigating the security of the masked implementations in different security models, starting from the most abstract "probing model" of Ishai et al. [27], following with the intermediate "bounded moment model" of Barthe et al. [5] and ending with the most concrete "noisy leakage model" of Prouff and Rivain [38], one can gradually build a confident assessment of the security level.

Finally, we apply our new multi-model methodology to our implementations of the AES and Fantomas, and discuss its limitations. Its application allows us to claim so far unreported security levels (e.g., against adversaries exploiting more than 2^{64} measurements) and to conclude that, in front of worst-case adversaries taking advantage of all the exploitable leakage samples in an implementation, performance improvements naturally lead to security improvements.

2 Background

In this section, we recall the parallel masking scheme we aim to study, and the two block ciphers we choose to work with, namely the AES and Fantomas.

2.1 Barthe et al.'s parallel masking algorithm

Masking is a popular side-channel countermeasure formalized by the seminal work of Ishai et al. [27]. Its main idea is to split all the key dependent data (often called sensitive variables) in different pieces which are randomly generated. More formally, masking consists in sharing a sensitive value s such that:

$$s = s_1 \oplus s_2 \oplus \cdots \oplus s_d.$$

In the case of Boolean masking we will consider next, $\oplus$ is the XOR operation, each share s_i is a random bit and d is the number of shares. In order to apply masking to a block cipher, one essentially needs a way to perform secure multiplications and to refresh the shares. In the case of the bitslice implementations

we will consider next, this amounts to perform secure AND gates and XORing with fresh random values. For this purpose, we will use the algorithms proposed by Barthe et al. at Eurocrypt 2017 [5]. Namely, and following their notations, we denote as $\boldsymbol{a} = (a_1, a_2, \cdots, a_d)$ a vector of d shares, by $\mathsf{rot}(\boldsymbol{a}, n)$ the rotation of vector $\boldsymbol{a}$ by n positions. Moreover, the bitwise addition and multiplication operations (i.e., the XOR and AND gates) between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are denoted as $\boldsymbol{a} \oplus \boldsymbol{b}$ and $\boldsymbol{a} \cdot \boldsymbol{b}$, respectively. Based on these notations, the refreshing algorithm is given by Algorithm 1 for any number of shares d. Its time complexity is constant in the number of shares d and requires d bits of fresh uniform randomness.

Algorithm 1 Parallel Refreshing Algorithm

Input: Shares $\boldsymbol{a}$ satisfying $\bigoplus_i a_i = a$, uniformly random vector $\boldsymbol{r}$
Output: Refreshed shares $\boldsymbol{b}$ satisfying $\bigoplus_i b_i = a$
$\boldsymbol{b} = \boldsymbol{a} \oplus \boldsymbol{r} \oplus \mathsf{rot}(\boldsymbol{r}, 1)$
return $\boldsymbol{b}$

For readability, we next give the multiplication algorithm for the case $d = 4$ in Algorithm 2. Its description for any d can be found in [5]. The time complexity of the algorithm is linear in the number of shares d and it requires $d \cdot \lceil \frac{d-1}{4} \rceil$ bits of randomness. Intuitively, this algorithm can be viewed as a combination of different steps: (1) the loading (and possible rotation) of the input share(s), (2) a partial product phase between the shares, (3) the loading and rotation of the fresh randomness, and (4) a compression phase where partial products are XORed together, interleaved with the addition of fresh randomness.

Algorithm 2 Parallel Multiplication Algorithm for $d = 4$

Input: Shares $\boldsymbol{a}$ and $\boldsymbol{b}$ satisfying $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$, unif. rand. vector $\mathbf{r}$
Output: Shares $\mathbf{x}$ satisfying $\bigoplus_i x_i = a \cdot b$
$\boldsymbol{c}_1 = \boldsymbol{a} \cdot \boldsymbol{b}$
$\boldsymbol{c}_2 = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, 1)$
$\boldsymbol{c}_3 = \mathsf{rot}(\boldsymbol{a}, 1) \cdot \boldsymbol{b}$
$\boldsymbol{d}_1 = \boldsymbol{c}_1 \oplus \mathbf{r}$
$\boldsymbol{d}_2 = \boldsymbol{d}_1 \oplus \boldsymbol{c}_2$
$\boldsymbol{d}_3 = \boldsymbol{d}_2 \oplus \boldsymbol{c}_3$
$\boldsymbol{d}_4 = \boldsymbol{d}_3 \oplus \mathsf{rot}(\mathbf{r}, 1)$
$\boldsymbol{x} = \boldsymbol{d}_4$
return $\boldsymbol{x}$

2.2 Target algorithms

The AES Rijjndael [13] is a 128-bit block cipher operating on bytes and allowing three different key sizes (128, 192 and 256 bits). We will focus on the 128-bit variant that has 10 rounds. Each round is composed of the succession of 4 operations: SubBytes (which is the non-linear part), ShiftRows, MixColumns and AddRoundKey (except for the last round where MixColumns is removed). Each round key is generated thanks to a key schedule algorithm. Operations will be

detailed in the implementation section. The AES' robustness over the years and widespread use makes it a natural benchmark to compare implementations.

Fantomas is an instance of LS-Design [23], of which the main goal is to make Boolean masking easy to apply. It is a 128-bit cipher iterating 12 rounds based on the application of an 8-bit bitslice S-box followed by a 16-bit linear layer (usually stored in a table and called the L-box), together with a partial round constant addition and a key addition. The internal state of Fantomas can be seen as an 8×16-bit matrix where the S-box is applied on the columns and the L-box is applied on the rows. The precise description of the S-box and L-box are provided in Appendix A, in Algorithm 3 and Figure 4, respectively.

We note that another instance of LS-design (namely Robin) has been recently cryptanalyzed by Leander et al. [29] and Todo et al. [47]: both attacks highlight a dense set of weak keys in the algorithm and can be thwarted by adding full round constants in each round [28]. Despite there is no public indication that a similar attack can be applied to Fantomas, we considered a similar tweak as an additional security margin (and denote this variant as Fantomas*).

2.3 Target device and measurement setups

Our implementations are optimized for a 32-bit ARM Cortex-M4 processor clocked at 100 MHz and embedded in the SAM4C-EK evaluation board [1]. Of particular interest for our experiments, this device has an embedded True Random Number Generator (TRNG) which provides 32 bit of randomness every 80 clock cycles. We recall the description of the ARM processor and instructions set given in [22]. The processor is composed of sixteen 32-bit purpose registers labeled from R0 to R15. Registers R0 to R12 are the variable registers (available for computations), R13 contains the stack pointer, R14 contains the link register and R15 is the program counter. The ARM instructions can be classified in three distinct sets: the data instructions such as AND, XOR, OR, LSR, MOV, ..., which cost 1 clock cycle; the memory instructions such as STR, LDR,..., which cost 2 clock cycles (with the thumb extension); and the branching instructions such as B, BL, BX, ..., which cost from 2 to 4 clock cycles. A useful property of the ARM assembly is the barrel shifter. It allows applying one of the following instructions on one of the operands of any data instruction for free: the logical shift (right LSR and left LSL), the arithmetic shift right ASR and the rotate-right ROR.

As for our security evaluations, we performed power analysis attacks using a standard setup measuring voltage variations across a resistor inserted in the supply circuit, with acquisitions performed using a Lecroy WaveRunner HRO 66 oscilloscope running at 625 Msamples/second and providing 8-bit samples.

3 Efficient implementations

We designed our implementations in a modular manner, starting with building blocks such as refreshing and multiplication algorithms, and then building more complex components such as the S-boxes, rounds, and full cipher upon the previous ones. This adds flexibility to the implementation (i.e., we can easily change

one of the building blocks, for example the random number generator) and enables simple cycle counts for various settings. Following this strategy, we first describe the implementation of cipher independent operations, and then discuss optimizations that specifically relate to the AES and Fantomas*.

3.1 Cipher independent components

We start by setting up the parameters of our parallel masking scheme and then depict the implementation of the refreshing and multiplication algorithms.

Given the register size r of a processor, parallel masking offers different trade-offs to store the shares of a masked implementation. In the following, we opted for the extreme solution where the number of shares d equals r (which minimizes the additional control overheads needed to store the shares of several intermediate values in a single register). In our 32-bit ARM processor example, this implies that we consider a masked implementation with 32 shares.

More precisely, let $\boldsymbol{s} = (s_1, \cdots s_{32})$ be a 32-bit word where each s_i for $1 \leq i \leq 32$ is a bit and s be a sensitive bit. We have that $s = \bigoplus_{i=1}^{32} s_i$. Concretely, our implementations will store such vectors of 32 shares corresponding to a single bit of sensitive data in single registers. This allows us to take advantage of the parallelization offered by bitwise operations such as XOR, AND, OR, ... That is, let $\perp$ be such a bitwise operator and $\boldsymbol{s}^a$, $\boldsymbol{s}^b$ two 32-bit words, we have:

$$\boldsymbol{s}^a \perp \boldsymbol{s}^b = (s_1^a \perp s_1^b, \cdots, s_{32}^a \perp s_{32}^b).$$

In practice, for a block cipher of size n with key size k, its internal state will therefore be represented and stored as $n+k$ 32-bit words in our parallel masking setting. The initial key sharing (performed once in a leak-free environment) is done as usual by ensuring that the s_i's are random bits for $2 \leq i \leq d$ and $s_1 = s \oplus s_2 \oplus \cdots \oplus s_d$. These shares are then refreshed with Algorithm 1 before each execution. And the un-sharing can finally be done by computing the value $\bigoplus_{i=1}^{32} s_i$, or equivalently by computing the Hamming weight modulo 2 of $\boldsymbol{s}$.

One natural consequence of this data representation is that it requires the block cipher description to be decomposed based on Boolean operations. Bitslice ciphers such as Fantomas* are therefore very suitable in this context, since directly optimized to minimize the complexity of such a decomposition.

Refreshing and Multiplication algorithms Since only requiring simple AND, XOR and rotation operations, these algorithms have naturally efficient implementations on our target device. The only particular optimization we considered is to keep all intermediate values in registers whenever possible, in order to minimize the overheads due to memory transfers. For illustration, Appendix B provides an ARM pseudo-code for the multiplication with $d = 4$. The random values needed for the refreshings are first loaded and kept in registers. We then compute the c_i's and d_i's together instead of successively as in Algorithm 2, allowing to save costly load and store instructions. Eventually, the randomness

was produced according to two different settings. In the first one, we generated it on-the-fly thanks to the embedded TRNG of our board which costs $RC = 80$ clock cycles per 32-bit word. In the second one, we considered a cheaper PRG following the setting of [22], which costs $RC = 10$ cycles per 32-bit word. Based on these figures, the refreshing algorithm is implemented in 28 (resp. 98) clock cycles and the multiplication algorithm in 197 (resp. 757) clock cycles.

3.2 Cipher dependent components

We now describe how we implemented the AES Rijndael and Fantomas* in bitslice mode rather than in based on their (more) usual byte representation.

AES components. The AES S-box is an 8-bit permutation which can be viewed as the composition of an inverse in $\mathbb{F}_{2^8}$ and an affine function. A well-known method to mask this S-box, first proposed by Rivain and Prouff in [42], is to decompose the inversion in a chain of squarings and multiplications. Yet, this decomposition is not convenient in our parallel masking setting since not based on binary operations. Hence, a better starting point for our purposes is the binary circuit put forward by Boyar and Peralta in 2010 [8]. It requires 83 XOR, 32 AND and 4 NOT gates. Recently, Goudarzi and Rivain re-arranged some operations of this circuit in order to improve their implementation of a masked bitsliced AES [22]. We therefore implemented the AES S-box thanks to the latter representation, with each AND replaced by a secure multiplication and the XORs transposed using the corresponding ARM assembly instructions.

Following, and thanks to our internal state representation, the ShiftRows operation is easy to implement: it just consists in a re-ordering of the data which is achieved by a succession of load and store instructions.

The AES MixColumns operation is slightly more involved. The usual representation of MixColumns is based on a matrix product in $\mathbb{F}_{2^8}$, as depicted in the following, where c_i and d_i for $0 \leq i \leq 3$ are bytes:

$$\begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \times \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}.$$

The multiplication by 01 is trivial and the one by 03 can be split into $02 \oplus 01$, which only leaves the need of a good multiplication by 02 (sometimes called the xtimes function). This function is usually performed thanks to pre-computed tables [13], but it can also be achieved solely with binary instructions. Let $b = (b_0, \cdots, b_7)$ be a byte with $b_i \in \{0, 1\}$ for $0 \leq i \leq 7$. We recall that the AES field is defined as $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. Using this polynomial, the xtimes can be turned into the following Boolean expression:

$$\mathsf{xtimes}(b) = \mathsf{xtimes}(b_0, \cdots, b_7) = (b_1, b_2, b_3, b_4 \oplus b_0, b_5 \oplus b_0, b_6, b_7 \oplus b_0, b_0). \quad (1)$$

For the parallel masking scheme, each bit b_i is again replaced by a 32-bit word. So in practice, we simply implement Mixcolumns by small pieces: for each byte c_i we load the eight 32-bit words, compute all the products by 02 thanks to Equation (1), and store the results in a temporary memory slot. Eventually, we recombine the temporary values by XORing them to obtain the right output.

Fantomas* components. Fantomas*'s 8-bit S-box is an unbalanced Feistel network built from 3- and 5-bit S-boxes originally proposed in the MISTY block cipher (see [34], Section 2.1 and [23]). It can be decomposed in 11 AND gates, 25 XOR gates and 5 NOT gates. Since the S-box is bitsliced, the implementation of the parallel scheme is straightforward. Namely, each W_i in Algorithm 3 is a 32-bit word encoding one secret bit in 32 shares. As for the AES S-box, ANDs are replaced by secure multiplications and XORs are applied directly.

The Fantomas* linear layer so-called L-box can be represented as a 16×16 binary matrix M (see Figure 4 in Appendix A). Let V a 16×1-bit vector of the internal state of Fantomas*. Applying the L-box consists in doing the product $M * V$, which corresponds to executing XOR gates between the bits of V, defined by the entries of the matrix M. Since the XOR is a bitwise and linear operation, the L-box can again be computed directly in the parallel masking context (where a bit in the vector V simply becomes a 32-bit word of shares). In practice, as in the original publication of Fantomas* [23], we split M in two 16×8 matrices: a left one and a right one. This allows us to work independently with the first 8 bits and the last 8 bits of V. For this purpose, we load eight 32-bit words and compute the XORs between them corresponding to the left/right parts of M, and store these intermediate values in a temporary memory slot. Eventually, one has just to XOR the results of these two products to recover the output.

3.3 Performance evaluation

Table 1 provides the total number of total clock cycles for both the AES and Fantomas* in our two settings for the randomness generation. The S-box column reports the percentage of clock cycles spent in the evaluation of the S-box (excluding the randomness generation and refreshings). The linear layer column reports the percentage of clock cycles spent in the evaluation of the linear parts (i.e., ShiftRows, MixColumns and AddRoundKey for the AES; the L-boxes, key and round constant additions for Fantomas*). The rand. column reports the percentage of clock cycles spent in the generation of fresh random numbers (in-

	total # of cycles	S-box %	linear Layer %	rand. %
AES ($RC = 10$)	2,783,510	25.16	1.99	72.66
AES ($RC = 80$)	9,682,710	7.23	0.6	91.91
Fantomas* ($RC = 10$)	1,217,616	23.95	4.6	68.56
Fantomas* ($RC = 80$)	4,134,096	7.06	1.38	90.74

Table 1. Performance evaluation results for $d = 32$.

cluding the refresh operations and random values needed in the multiplication). Note that in order to make our results comparable with the ones of Goudarzi and Rivain, we did not consider the evaluation of the AES key schedule and simply assumed that the round keys (or the master key for Fantomas*) were pre-computed, stored in a shared manner and refreshed before each execution of the ciphers. Besides, and as in this previous work (Section 6.2), we systematically refreshed one of the inputs of each multiplication in order to avoid flaws related to the multiplication of linearly-related inputs.[1] The masked AES implementation in [22] is evaluated on a device similar to ours with up to 10 shares. Using their cost formulas, we can extrapolate the number of clock cycles of their implementation for $d = 32$ shares as approximately $3,821,312$ cycles (considering $RC = 10$), which highlights that the linear complexity of our multiplication algorithm indeed translates into excellent concrete performances. The further comparison of our (share-based) bitslicing approach with the (algorithm-based) one in [22] is an interesting scope for further research. In this respect, we note the focus of our codes was on regularity and simplicity, which allowed fast development times while also leaving room for further optimizations.

As expected, using the bitslice cipher Fantomas* rather than the standard AES Rijndael allows reducing the cycle counts by an approximate factor 2. This is essentially due to the fact that the overall number of secure multiplications of the latter is roughly twice lower (2112 against 5120 multiplications).

This benchmarking highlights that the time spent in the linear layers in very high order (parallel) masked implementations is negligible: efforts are spent in the S-box executions and (mostly) the randomness generation. It suggests various tracks for improved designs, ranging from the minimization of the non-linear components thanks to powerful linear layers, the reduction of the randomness requirements in secure multiplications or the better composition of linear & non-linear gadgets (see Sections 4.1 and 4.3), and the design of efficient RNGs.

4 Side-channel security evaluation

The previous section showed that bitslice implementations of masking schemes lead to excellent performances, as already hinted by Goudarzi and Rivain [22], and that the parallel refreshing and multiplication algorithms of Barthe et al. in [5] are perfectly suited to them. Thanks to these advances, we are able to obtain realistic timings for very high order masked implementations.

Quite naturally, such very high order implementations raise the complementary challenge that they are not trivial to evaluate. In particular, since one can expect that they lead to very high security levels (if their shares' leakages are independent and sufficiently noisy), an approach based on "launching attacks"

[1] We used the iteration of $\lceil \frac{d-1}{3} \rceil$ simple refreshing gadgets (given Algorithm 1) for this purpose, which is conjectured to be composable in [5] (and therefore comparable to the refreshing used in [22]). As will be discussed in Sections 4.1 and 4.3, this very direct strategy leaves ample room for further optimization efforts.

is unlikely to provide any meaningful conclusion. That is, unsuccessful attacks under limited evaluation time and cost do not give any indication of the actual security level (say 2^x) other than that the evaluator was unable to attack in complexity 2^y, with potentially $2^x \gg 2^y$. In the following, we introduce a new methodology for this purpose, based on recent progresses in the formal analysis of masking exploiting different proof techniques and leakage models.

4.1 Rationale: a multi-model approach

The core idea of our following security evaluation is to exploit a good separation of duties between the different leakage models and metrics that have been introduced in the literature. More precisely, we will use the probing model of Ishai et al. to guarantee an "algorithmic security order" [27], the bounded moment model of Barthe et al. to guarantee a "physical security order" [5], and the noisy leakage model of Prouff and Rivain to evaluate concrete security levels [38].

Step 1. The probing model, composability and formal methods. In general, the first important step when evaluating a masked implementation is to study its security against (abstract) t-probing attacks. In this model, the adversary is able to observe t wires within the implementation (usually modeled as a sequence of operations). From a theoretical point of view, it has been shown in [14] that (under conditions of noise and independence considered in the following steps), probing security is a necessary condition for concrete (noisy leakage) security against (e.g., power or electromagnetic) side-channel attacks. It has also been shown in [5] that it is equally relevant in the case of parallel implementation we study here (i.e., that it is also a necessary condition in this context).

From a practical point of view, the probing security of simple gadgets such as given by Algorithms 1 and 2 is given in their original papers, and the main challenge for their application to complete ciphers is their composability. That is, secure implementations must take into account the fact that using the output of a computational gadget (e.g., an addition or multiplication) as the input of another computational gadget may provide additional information to the adversary. Such an additional source of leakage is essentially prevented by adding refreshing gadgets. There exists two strategies to ensure that the refreshings in an implementation are sufficient. First, one can use probing-secure computational gadgets, test implementations with formal methods such as [3], and add refreshing gadgets whenever a composition issue is spotted by the tool. This solution theoretically leads to the most efficient implementations, but is limited by the complexity of analyzing full implementations at high orders. Second, one can impose stronger (local) requirements to the computational gadgets, such as the Strong Non Interference (SNI) property introduced in [4]. Those gadgets are generally more expensive in randomness, but save the designers/evaluators from the task of analyzing their implementation globally. As mentioned in Section 3.3 we exploited a rough version of this second strategy, by applying an SNI refreshing to one input of every multiplication. As discussed in [7] (e.g., when masking

the AES S-box based on a polynomial representation in Section 7.2), it is actually possible to obtain SNI circuits with less randomness thanks to a clever combination of SNI and NI gadgets. The investigation of such optimizations in the case of bitslice implementations is an interesting open problem.

Step 2. The bounded moment model and Welch's T-test. Given that probing security is guaranteed for an implementation, the next problem is to guarantee the shares' leakages physical independence. In other words, the evaluator needs to test whether the leakage function does "re-combine" the shares in some way that is not detectable by abstract probing attacks. From a theoretical viewpoint, this recombination can be captured by a reduction of the security order in the bounded moment model [5]. Concretely, it may be due to defaults such as computational glitches [31, 32] and memory transitions [11, 2].

From a practical point of view, the security order in the bounded moment leakage model can be estimated thanks to so-called "moment-based security evaluations". One option for this purpose is to launch high order attacks such as [49, 39, 35]. In recent years, and alternative and increasingly popular solution for this purpose has been to exploit simple(r) leakage-detection tests [21, 10, 33, 44, 16]. We will next rely on the recent discussion and tools from [46].[2]

Step 3. The noisy leakage model and concrete evaluations. Eventually, once a designer/evaluator is convinced that his target implementation guarantees a certain security order, it remains to evaluate the amount of noise in the implementation. Indeed, from a theoretical point of view, a secure masking scheme is expected to amplify the impact of the noise in any side-channel attack (and therefore the worst-case measurement complexity) exponentially in the number of shares. This concrete security is reflected by the noisy leakage model [38].

From a practical point of view, the noise condition for secure masking (and the resulting noisy leakage security) can be captured by an information theoretic or security analysis [45]. In this respect, it is important to note that this condition depends on both the physical noise of the operations in the target implementation and the number of such operations. When restricting the evaluation to divide-and-conquer attacks, which is the standard strategy to exploit physical leakages [30], this number of operations drops to the number of exploitable operations (i.e., the operations that depend on an enumerable part of the key). We will next consider this standard adversarial setting.[3]

Besides, as mentioned at the beginning of the section, one may expect that the security level of a very high order masked implementation is beyond the evaluator's measurement (and time, memory) capacities. In this context, rather than trying to launch actual attacks we will rely on the (standard cryptographic)

[2] Note that nothing prevents using the bounded moment model to analyze abstract implementations: as shown in [5] it may also allow explaining the security of certain types of countermeasures that cannot be captured by probing security.

[3] More advanced strategies include algebraic/analytical side-channel attacks, which may require considering slight additional (constant) security margins [41, 48, 24].

strategy of bounding the attack complexity based on the adversary's power. For this purpose, we will use the tools recently introduced in [15, 26] which show that such bounds can be obtained from the information theoretic analysis of the leakage function (i.e., a characterization of the individual shares' leakages).

Wrapping up. The main observation motivating our rationale is that security against side-channel attacks can be gradually built by exploiting existing leakage models, starting from the most abstract probing model, following with the intermediate bounded moment model, and finishing with the most physical noisy leakage model. In this respect, one great achievement of recent research in side-channel analysis is that each of those theoretical leakage models has a concrete counterpart allowing its practical evaluation. Namely, the probing security of an algorithm (represented as a sequence of operations) is challenged by formal methods or guaranteed by composable gadgets, bounded moment security is tested thanks to moment-based distinguishers or leakage-detection tools, and noisy leakage security is quantified thanks to information theoretic metrics which eventually bound standard security metrics such as the success rate.

Cautionary note. Because of place constraints, the following sections will not recall the technical details of the tools used in our evaluations (i.e., Welch's T-test, linear regression and the mutual information metric). We rather specify all the parameters used and link to references for the description of the tools.

4.2 Bounded moment security and security order

Noise-efficient leakage detection test. As we rely on SNI refreshings to ensure the composability of our masked implementations, the first step in our evaluation is to evaluate the extent to which the shares' physical leakages are independent.[4] As mentioned in the previous subsection, this independence is reflected by a security order in the bounded moment model, which can be estimated thanks to leakage detection. For this purpose, we used a variant of leakage detection test recently introduced in [46], Section 3.2. As with the standard detection tools, its main idea is to consider two leakage classes: one corresponding to a fixed plaintext and key, the other corresponding to random (or fixed [16]) plaintext(s) and a fixed key. The test then tries to detect a differences between these classes at different orders (i.e., after raising the leakage samples to different powers). The only specificity of this "noise-efficient" variation is that it mitigates the exponential amplification of the noise due to masking by averaging the traces before raising them to some power, thus reducing the evaluation time and storage. Such an averaging is possible because of our evaluation setting where masks are known. It admittedly makes the test completely qualitative (i.e., the number of traces needed to detect is not correlated with the security level that we discuss in the next subsection). Yet, in view of the noise level of our implementation, it was the only way to detect high order leakages somewhat efficiently.

[4] Analyzing the SNI security of a software code (rather than an abstract implementation as usually done in masking gadget proofs) would further increase the relevance of the composability argument and is an interesting scope for further research.

Unfortunately, and even using this tweak, the complexity of the leakage detection is still exponential in the number of shares and therefore hardly achievable at order 32 (see again [46]). As a result, we studied reduced-order implementations with limited number of shares/randomness. Similarly to reduced-round versions in block cipher cryptanalysis, the goal of such implementations is to extrapolate the attacks' behavior based on empirically verifiable but weakened versions of our implementations. In particular, we used such implementations to verify the extent to which the shares are recombined by the physical leakages. Since the implementations considered for this purpose are similar to the one using 32 shares (see next), the hope is that they give the evaluator an estimate of the "security order reduction factor" f caused by physical defaults (e.g., [2] showed that transition-based leakages reduce this order by a factor two).

Concretely, we analyzed both tweaked implementations with $d = 2$ and $d = 4$ shares (thanks to an adapted software) and the implementation with 32 shares where only 2 (resp. 4) bits of the random numbers generated were actually random – the other 30 (resp. 28) bits being kept constant. All tests gave consistent results and no leakage of order below the expected 2 (resp. 4) was detected. For illustration, the result of a leakage detection test for the $\mathsf{Fantomas}^*$ S-box with $d = 4$ shares (tweaked implementation) is given in Figure 1. We used 120,000 different traces, each of them repeated 50 times, for a total of 6,000,000 measurements. The top of the figure shows the average trace, the bottom of the figure is the result of the detection test at order 4, where we see that the standard threshold of 4.5 is passed for a couple of samples. We additionally checked that those samples indeed correspond to the multiplications performed during the S-box execution. By contrast, Figure 5 in Appendix C shows no evidence of detected leakage at lower orders. We insist that testing such reduced-order implementations does admittedly not offer formal guarantees that no flaw may happen for the full version with 32 random shares.[5] Nevertheless, (*i*) the fact that we observed consistent results for the $d = 2$ and $d = 4$ cases is reassuring; (*ii*) we may expect that some physical defaults (such as couplings [9]) become less critical with larger number of shares, since the shares will be more physically separated in this case; and (*iii*) most importantly, we will use the factor f as a parameter of our security evaluations, allowing a good risk assessment.

Robustness against transition-based leakages. The results of the previous detection tests are (positively) surprising since one would typically expect that the transition-based leakages discussed in [2] reduce the security order in the bounded moment model from the optimal $o{=}d-1$ to $o{=}\lceil d/2 \rceil - 1$. For example, assuming a sharing $s = s_1 \oplus s_2$, observing the Hamming distance between the shares s_1 and s_2 would provide the adversary with leakages of the form $\mathsf{HD}(s_1, s_2) = \mathsf{HW}(s_1 \oplus s_2) = s$. By contrast, in our parallel implementation setting, no such transitions could be detected. While we leave the full analysis of this phenomenon (e.g., with formal methods) as an open problem, we next provide

[5] Note also that the variant of leakage detection with averaging used here makes the interpretation of such flaws less easy to interpret with the tools of [15] (Section 4.2).

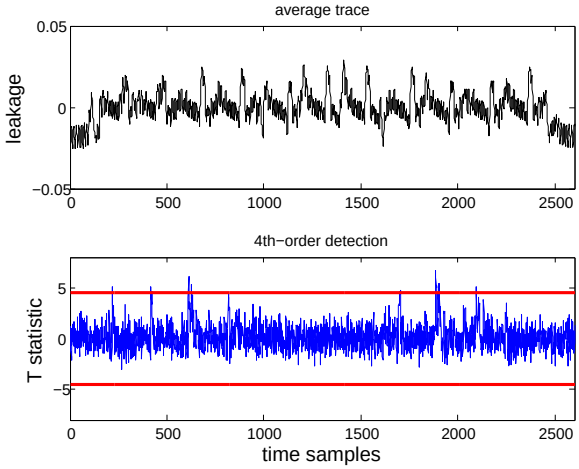

Fig. 1. Noise-efficient leakage detection with 6M traces (50x averaging).

preliminary explanations why this positive result is at least plausible. For this purpose, we first observe that the multiplication Algorithm 2 essentially iterates three types of operations: partial products, compressions and refreshings; and it ensures that any pair of partial products $(a_i \cdot b_j,\ a_j \cdot b_i)$ is separated from the other pairs (and the $a_i \cdot b_i$ partial products) by a refreshing. As already hinted in [5], the distances between such pairs of intermediate results do not lead to additional information to the adversary. So the main source of transition-based leakages would be based on intermediate results separated by refreshings. In this respect, we note that our implementation was designed so that intermediate results are produced progressively according to the previous "compute partial products – compress – refresh" structure, which additionally limits the risk that many unrefreshed intermediates remain in the registers. Eventually, we checked that intermediate results in successive clock cycles do not lead to detectable transition-based leakages in the bounded moment model thanks to simulations. So intuitively, we can explain the absence of such transition-based leakages by the fact that our parallel manipulation of the shares mitigates them.[6]

Summarizing, as any hypothesis test, leakage detection offers limited theoretical guarantees that no lower-order leakages could be detected with more measurements. Yet, our experiments do not provide any evidence of strong recombinations of the shares' leakages via transitions or other physical defaults, which can be explained by algorithmic features. Hence, in the following, we will consider two possible settings for our evaluations: the empirically observed one, assuming a security order 31 in the bounded moment model, and a more conservative one, assuming a security order 15 in the bounded moment model.

[6] When decreasing technology sizes, this gain may come with higher risk of couplings between the shares (as also mentioned in [5] and recently discussed in [9]).

4.3 Noisy leakage security and information theoretic analysis

Assuming the security order of our implementations to be 31 (as observed experimentally) or 15 (taking a security margin due to a risk of physical defaults that we could not spot), we now want to evaluate the security level of these implementations in the noisy leakage model, based on an information theoretic and security analysis. For this purpose, our next investigations will follow three main steps. First we will estimate the deterministic and noisy parts of the leakage function corresponding to our measurements, thanks to linear regression [43]. This will additionally lead to an estimation of our implementations' Signal to Noise Ratio (SNR). Second, we will use this estimation of the leakage function to quantify the information leakage of our Boolean encodings (assuming security orders 31 and 15, as just motivated), using the numerical integration techniques from [15]. Finally, we will take advantage of the tightness of masking security proofs recently put forward in [26], in order to bound the complexity of multivariate (aka horizontal) attacks taking advantage of all the leakage samples computationally exploitable by a divide-and-conquer side-channel adversary.

Linear regression and noise level. For this first step, we again considered a simplified setting where the evaluator has access to the masks during his profiling phase. Doing so, he is able to efficiently predict the 32 bits of the bus in our ARM Cortex device, and therefore to estimate the leakage function for various target operations thanks to linear regression. More precisely, and given a sensitive value s and its shares vector $\boldsymbol{s}$ considered in our masking scheme, linear regression allows estimating the true leakage function $\hat{\mathsf{L}}(\boldsymbol{s}) \approx \hat{\mathsf{D}}(\boldsymbol{s}) + \hat{N}$, with $\hat{\mathsf{D}}(\boldsymbol{s})$ the deterministic part of the leakages and $\hat{N}$ a noise random variable. As frequently considered in the literature, we used a linear basis (made of the 32 bits of the bus and a constant element) for this purpose. Such a model rapidly converged towards close to Hamming weight leakages, with estimated SNR of 0.05 for the best sample (defined as the variance of $\hat{\mathsf{D}}(\boldsymbol{s})$ divided by the variance of $\hat{N}$).

Encoding leakage. Given the previous sensitive value s, its shares vector $\boldsymbol{s}$ considered in our masking scheme, and a leakage function L leading to samples $l = \mathsf{L}(\boldsymbol{s})$, a standard metric to capture the informativeness of these leakages is the Mutual Information [45], defined as follows:

$$\mathrm{MI}(S; \mathsf{L}(\boldsymbol{S})) = \mathrm{H}[S] + \sum_{s \in \mathcal{S}} \Pr[s] \cdot \sum_{l \leftarrow \mathsf{L}} \mathsf{f}(l|s) \cdot \log_2 \Pr[s|l].$$

In this equation, $\mathrm{H}[S]$ is the entropy of the sensitive variable S and $\mathsf{f}(l|s)$ the conditional Probability Density Function (PDF) of the leakages $\mathsf{L}(\boldsymbol{s})$ given the secret s. Assuming Gaussian noise, it can be written as a mixture model:

$$\mathsf{f}(l|s) = \sum_{\boldsymbol{s} \in \mathcal{S}^{d-1}} \mathcal{N}\left(l|(s, \boldsymbol{s}), \sigma_n^2\right) \cdot$$

The conditional probability $\Pr[s|l]$ is then computed thanks to Bayes' theorem as:

$$\Pr[s|l] = \frac{\mathsf{f}(l|s)}{\sum_{s^* \in \mathcal{S}} \mathsf{f}(l|s^*)}.$$

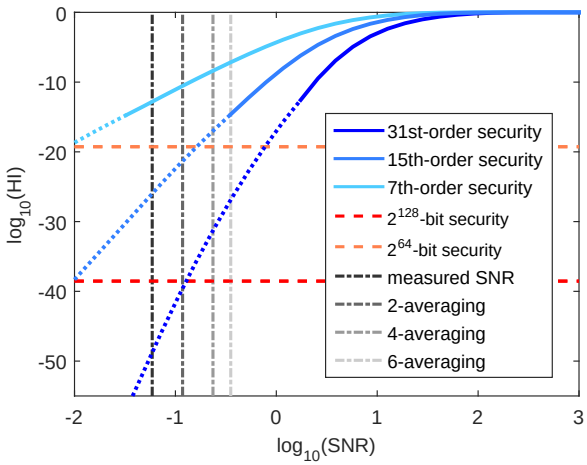

Fig. 2. Information theoretic analysis of the encoding.

Unfortunately, what we obtained thanks to linear regression is not the true leakage function $\mathsf{L}(\boldsymbol{s})$ but only its estimate $\hat{\mathsf{L}}(\boldsymbol{s})$. Hence, what we will compute in the following is rather the Hypothetical Information (HI), defined as:

$$\mathrm{HI}(S; \hat{\mathsf{L}}(\boldsymbol{S})) = \mathrm{H}[S] + \sum_{s \in \mathcal{S}} \Pr[s] \cdot \sum_{l \leftarrow \hat{\mathsf{L}}} \hat{\mathsf{f}}(l|s) \cdot \log_2 \hat{\Pr}[s|l].$$

Formally, it corresponds to the amount of information that would be leaked from an implementation of which the leakages would be exactly predicted by $\hat{\mathsf{L}}(\boldsymbol{s})$. Admittedly, we cannot expect that $\mathrm{HI}(S; \hat{\mathsf{L}}(\boldsymbol{S})) = \mathrm{MI}(S; \mathsf{L}(\boldsymbol{S}))$ in practice (e.g., since we used a linear basis rather than a full one in our regression).[7] However, we note that the information leakages of a masked implementation depend only on their security order and SNR, not on variations of the leakage function's shape. So small errors on $\hat{\mathsf{L}}$ should not affect our conclusions. Furthermore, in our parallel setting the addition of significant non-linear terms in the regression basis would also directly decrease the security order because it would re-combine the shares in a non-linear manner (see [5]). Since the previous moment-based evaluation did not detect such re-combinations, a linear leakage model is also well motivated from this side. We finally note that adding quadratic terms in our basis could be a way to capture the reduction of the security order from 31 to 15. Yet, for efficiency, we next reflect such reductions of the security order by simply (and pessimistically) reducing the number of random shares in $\boldsymbol{s}$.

The result of such an information theoretic evaluation for our Boolean encoding is given in Figure 2, where we plot the HI in log scale, for various SNRs. Of particular interest are the measured SNR and the SNRs with (2, 4 and 6×) averaging, which would correspond to the noise level of sensitive shares vectors

[7] Yet, we can test that it is close enough thanks to leakage certification [18, 17].

appearing multiple times in the implementation, therefore allowing the adversary to reduce the noise of these leakage samples by averaging (which we will discuss next). We also plotted the curves corresponding to the security orders 31, 15 and 7 (i.e., corresponding to a flaw parameter $f = 1, 2$ and 4). Remarkably, we see that for the measured SNR, the leakage of a single encoding secure of order 31 would lead to an HI below 2^{-128}. Since the masking proofs in [15] show that the measurement complexity of any side-channel attack is inversely proportional to (and bounded by) this information leakage, it implies that a simple attack exploiting a single leakage sample corresponding to a 32-tuple of parallel shares would not be successful even with the full AES/Fantomas* codebook. Similarly, a 15th-order secure implementation would be secure with up to a comfortable $10^{26} \approx 2^{82}$ measurements. Table 2 provides an alternative view of these findings and lists experimental HI values for different levels of averaging.

SNR	measured	×2	×3	×4	×5	×6	×7
Security order 31	-48	-39	-34	-31	-29	-27	-25
Security order 15	-26	-22	-19	-17	-16	-15	-14

Table 2. Experimental bounds on $\log_{10}$(HI) for the encoding.

Worst-case security level. While the previous figure and table show that an adversary exploiting a single 32-tuple of parallel shares, assuming security order 31 (or 15) and the SNR estimated in the previous section, will not be able to perform efficient key recoveries, it has been recently put forward in [6] and more formally discussed in [26] that optimal side-channel adversaries are actually much more powerful. Namely, such adversaries can theoretically exploit all the 32-tuples in the implementation, and if some of these tuples are manipulated multiple times, average their leakages in order to improve their SNR.

In order to take such a possibility into account in our security evaluations, we therefore started by inspecting the codes of our implementations in order to determine (1) the number of linear and non-linear operations that can be targeted by a divide-and-conquer attack (for illustration, we considered an adversary targeting a single S-box), and (2) the number of such operations for which one of the operands is repeated x times in the code. The result of such a code inspection is given in Table 3. Note that the table includes the count of the

Cipher	Operations	total #	2-rep.	3-rep.	4-rep.	5-rep.	6-rep.	7-rep.
AES	linear	115	20	55	18	12	10	0
	non-lin.	32	2	16	2	7	5	0
Fantomas*	linear	41	13	18	10	0	0	0
	non-lin.	11	1	5	5	0	0	0

Table 3. S-box code inspection for the AES and Fantomas*.

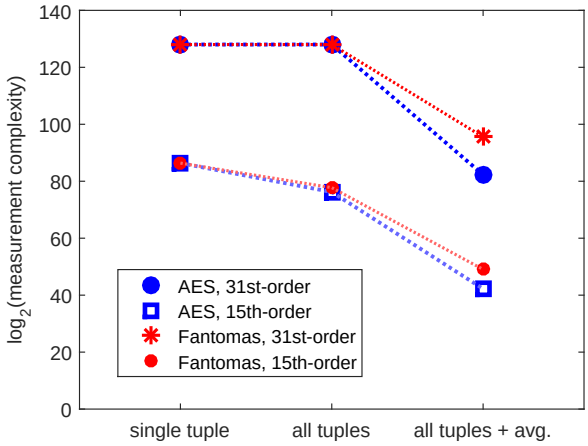

Fig. 3. Measurement complexity bounds for different attacks.

SNI refreshings added to one input of each multiplication, which we reported as 32 (resp. 11) additional linear operations for the AES (resp. Fantomas*).[8]

Thanks to the tools in [26], we then bounded the measurement complexity of adversaries taking advantage of a single tuple (considered in the previous section), all the tuples, and all the tuples with averaging in Figure 3. Concretely, the second adversary is simply captured by relying on an "Independent Operation Leakage" assumption which considers (pessimistically for the designer) that the information of all the 32-tuples of shares in the implementation is independent and therefore can be summed. Taking the example of the Fantomas* S-box, it means that this adversary can exploit the information of 41 encodings for the linear operations, and 11*32 encodings for the non-linear ones (where the factor 32 comes from the linear cost of the parallel multiplication algorithm, of which the leakage was bounded in [38]). And the third adversary is captured by adapting the encoding leakages depending on the number of repetitions allowed by the code. Taking the example of the linear operations in Fantomas*, it means that this adversary can exploit the information of 13 encodings with double SNR, 18 encodings with triple SNR, ... The latter is admittedly pessimistic too since it considers an averaging based on the most repeated operand only. Besides, it assumes that sensitive values manipulated multiple times will leak according to the same model (which is not always the case in practice [19]). The main observations of this worst-case security evaluation are threefold:

[8] This assumes that the iteration of simple refreshing gadgets to obtain an SNI refreshing is tweaked so that the tuple of shares to refresh is only XORed once, at the end of the iteration. It therefore slightly differs from the proposal in [5]. We leave the investigation of such a variant as an interesting scope for further research.

First, the security levels reached for the two first adversaries are significantly higher than previously reported thanks to "attack-based evaluations". In particular, we reach the full codebook (measurement) security if the security order was 31 (as empirically estimated) and maintain $> 2^{64}$ measurement security if this order was only 15. In this respect, we insist that this order is the *only* parameter which could lead to an overstated security level (i.e., all the other assumptions in our evaluations are pessimistic for the designer). Quite naturally, the figure also exhibits that masked implementations with lower orders (e.g., 8 or 4) cannot offer strong security guarantees in case of SNRs in the 0.01 range.

Second, the impact of averaging is much more critical, since the adversary then essentially cancels the exponential increase of the noise that is the expected payload of the masking countermeasure. Roughly, for an implementation secure of order d, doubling the SNR thanks to 2-averaging reduces the security by an approximate factor 2^d. By contrast, multiplying the number of target d-tuples (without averaging) by α only reduces the security by a factor α.

Third, in front of these optimal adversaries, Fantomas* offers (slightly) more security than the AES despite we assume the same information leakages for their encodings. This gain is essentially due to the fact that Fantomas* implementations are slightly more efficient, effectively reducing the opportunities for the adversary to exploit many leakage samples and to average them.

Towards mitigating averaging attacks. As a conclusion of this paper, we first observe that our experiments raise interesting optimization problems for finding new representations of block cipher S-boxes, minimizing the number of non-linear operations and the multiple manipulation of the same intermediate values during their execution. Besides, and quite fundamentally, Figure 3 recalls that the security of the masking countermeasure is the result of a tradeoff between an amount of physical noise (reflected by the SNR) and an amount of digital noise (reflected by the shares' randomness) in the implementations. In this respect, there is a simple way to mitigate the previous "averaging attacks", namely to add refreshing gadgets to prevent the repetition of the same sensitive values multiple times in an implementation. Remarkably, the systematic refreshing that we add to one input of each multiplication does contribute positively to this issue. For example, Table 4 in Appendix C shows that the number of repetitions in our codes increases if one removes these refreshings. By extending this approach brutally (i.e., by refreshing all the intermediate tuples in an implementation so that they are never used more than twice: once when generated, once when used), one can therefore mitigate the "all tuples + avg." adversary of Figure 3. But most interestingly, the latter observations suggest the search for good tradeoffs between physical and digital noise as a fundamental challenge for sound masking. That is, how to efficiently ensure composability as mentioned in Section 4.1 (first step) and prevent the averaging attacks in this section?

Acknowledgments. François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the INNOVIRIS project SCAUT and by the European Commission through the ERC project 724725 and the H2020 project REASSURE.

References

1. http://www.atmel.com/tools/sam4c-ek.aspx.
2. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer, 2014.
3. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Oswald and Fischlin [37], pages 457–485.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM, 2016.
5. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Coron and Nielsen [12], pages 535–566.
6. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Gierlichs and Poschmann [20], pages 23–39.
7. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 616–648. Springer, 2016.
8. Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *SEA 2010*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.
9. Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? Cryptology ePrint Archive, Report 2016/1080, 2016. `http://eprint.iacr.org/2016/1080`.
10. Jeremy Cooper, Elke De Mulder, Gilbert Goodwill, Josh Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice (extended abstract). ICMC 2013. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`.
11. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *COSADE 2012*, volume 7275 of *LNCS*, pages 69–81. Springer, 2012.
12. Jean-Sébastien Coron and Jesper Buus Nielsen, editors. *EUROCRYPT 2017*, volume 10210 of *LNCS*, 2017.
13. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
14. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Nguyen and Oswald [36], pages 423–440.
15. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Oswald and Fischlin [37], pages 401–429.

16. François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 240–262. Springer, 2016.
17. François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo. Towards easy leakage certification. In Gierlichs and Poschmann [20], pages 40–60.
18. François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to certify the leakage of a chip? In Nguyen and Oswald [36], pages 459–476.
19. Benoît Gérard and François-Xavier Standaert. Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version. *J. Cryptographic Engineering*, 3(1):45–58, 2013.
20. Benedikt Gierlichs and Axel Y. Poschmann, editors. *CHES 2016*, volume 9813 of *LNCS*. Springer, 2016.
21. Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. `http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf`.
22. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Coron and Nielsen [12], pages 567–597.
23. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 18–37. Springer, 2014.
24. Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 291–312. Springer, 2015.
25. Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: how large is the gap for AES? *J. Cryptographic Engineering*, 4(1):47–57, 2014.
26. Vincent Grosso and Franois-Xavier Standaert. Masking proofs are tight (and how to exploit it in security evaluations). Cryptology ePrint Archive, Report 2017/116, 2017. `http://eprint.iacr.org/2017/116`.
27. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
28. Anthony Journault, François-Xavier Standaert, and Kerem Varici. Improving the security and efficiency of block ciphers based on LS-designs. *Des. Codes Cryptography*, 82(1-2):495–509, 2017.
29. Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In Oswald and Fischlin [37], pages 254–283.
30. Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
31. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
32. Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Rao and Sunar [40], pages 157–171.

33. Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? An a priori statistical power analysis of leakage detection tests. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013*, volume 8269 of *LNCS*, pages 486–505. Springer, 2013.
34. Mitsuru Matsui. New block encryption algorithm MISTY. In Eli Biham, editor, *FSE '97*, volume 1267 of *LNCS*, pages 54–68. Springer, 1997.
35. Amir Moradi and François-Xavier Standaert. Moments-correlating DPA. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, pages 5–15, New York, NY, USA, 2016. ACM.
36. Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014*, volume 8441 of *LNCS*. Springer, 2014.
37. Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015*, volume 9056 of *LNCS*. Springer, 2015.
38. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.
39. Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
40. Josyula R. Rao and Berk Sunar, editors. *CHES 2005*, volume 3659 of *LNCS*. Springer, 2005.
41. Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: why time also matters in DPA. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 97–111. Springer, 2009.
42. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
43. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Rao and Sunar [40], pages 30–46.
44. Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptographic Engineering*, 6(2):85–99, 2016.
45. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.
46. Franois-Xavier Standaert. How (not) to use Welch's t-test in side-channel security evaluations. Cryptology ePrint Archive, Report 2017/138, 2017. `http://eprint.iacr.org/2017/138`.
47. Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear invariant attack - practical attack on full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10032 of *LNCS*, pages 3–33, 2016.
48. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 282–296. Springer, 2014.
49. Jason Waddle and David Wagner. Towards efficient second-order power analysis. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004.

A Fantomas* components

Algorithm 3 Fantomas* S-box.

Input: 8 words $(W_0, \ldots, W_7)$

First 5-bit S-box

$W_2 = W_2 \oplus (W_0 \wedge W_1)$
$W_1 = W_1 \oplus W_2$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_2 = W_2 \oplus W_3$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_4 = W_4 \oplus W_1$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_1 = W_1 \oplus W_0$

Extend-Xor

$W_0 = W_0 \oplus W_5$
$W_1 = W_1 \oplus W_6$
$W_2 = W_2 \oplus W_7$

Constant

$W_3 = \neg(W_3)$
$W_4 = \neg(W_4)$

First 3-bit S-box

$(t_5, t_6, t_7) = (W_5, W_6, W_7)$
$W_5 = W_5 \oplus \neg(t_6) \wedge t_7$
$W_6 = W_6 \oplus \neg(t_7) \wedge t_5$
$W_7 = W_7 \oplus \neg(t_5) \wedge t_6$

Truncate-Xor

$W_5 = W_0 \oplus W_5$
$W_6 = W_1 \oplus W_6$
$W_7 = W_2 \oplus W_7$

Second 5-bit S-box

$W_2 = W_2 \oplus (W_0 \wedge W_1)$
$W_1 = W_1 \oplus W_2$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_2 = W_2 \oplus W_3$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_4 = W_4 \oplus W_1$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_1 = W_1 \oplus W_0$

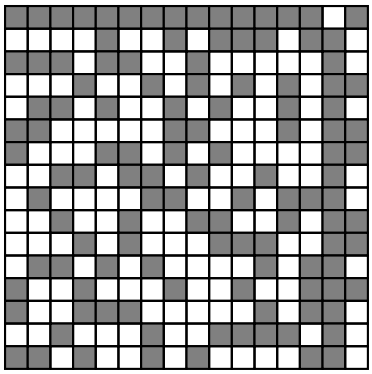

Fig. 4. Fantomas* L-box: dark cells stand for 1 and white cells stand for 0.

B ARM pseudo-code for a multiplication with $d = 3$

```
//r9 = a, r10 = b // r1 = random value
mov r2, #0xF //mask for rotation
//First computation
and r11, r9, r10 // c1 = a.b
eor r12, r11, r1 //d1 = c1 + r1
//rotation
lsr r3, r10, #1
lsl r4, r10, #3
orr r5, r3, r4
and r6, r5, r2 //r6 = rot(b,1)
and r11, r9, r6 //c2 = a.rot(b,1)
eor r12, r12, r11 // d2 = d1 + c2
//rotation
lsr r3, r9, #1
lsl r4, r9, #3
orr r5, r3, r4
and r6, r5, r2 //r6 = rot(a,1)
and r11, r10, r6 //c3 = b.rot(a,1)
eor r12, r12, r11 // d3 = d2 + c3
//rotation
lsr r3, r1, #1
lsl r4, r1, #3
orr r5, r3, r4
and r6, r5, r2 //r6 = rot(r1,1)
eor r12, r12, r6 //d4 = d3 + rot(r1,1)
//rotation
lsr r3, r10, #2
lsl r4, r10, #2
orr r5, r3, r4
and r6, r5, r2 //r6 = rot(b,2)
and r11, r9, r6 // c4 = a.rot(b,2)
eor r12, r12, r11 // x = d4 + c4
```

C Additional figures & tables

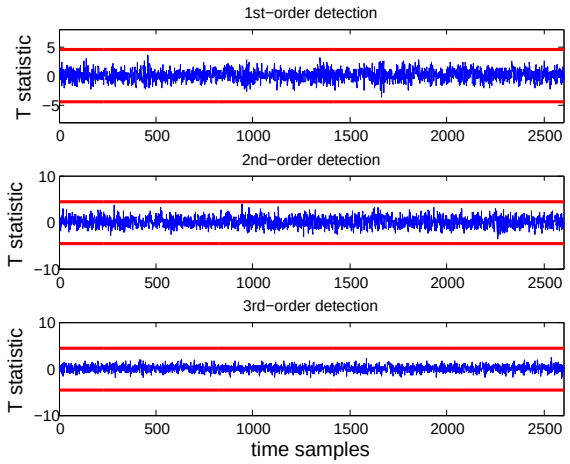

Fig. 5. Noise-efficient leakage detection with 6M traces (100x averaging).

Cipher	Operations	total #	2-rep.	3-rep.	4-rep.	5-rep.	6-rep.	7-rep.
AES	linear	83	7	43	7	15	7	4
	non-lin.	32	1	3	6	13	7	2
Fantomas*	linear	29	4	6	8	11	0	0
	non-lin.	11	1	3	2	5	0	0

Table 4. S-box code inspection for the AES and Fantomas* similar to Table 3, but without SNI refreshing at one input of each of their multiplications.