Masking vs. Multiparty Computation: How Large is the Gap for AES?

Vincent Grosso[1], François-Xavier Standaert[1], Sebastian Faust[2]

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland.

Abstract. In this paper, we evaluate the performances of state-of-the-art higher-order masking schemes for the AES. Doing so, we pay a particular attention to the comparison between specialized solutions introduced exclusively as countermeasures against side-channel analysis, and a recent proposal by Roche and Prouff exploiting MultiParty Computation (MPC) techniques. We show that the additional security features this latter scheme provides (e.g. its glitch-freeness) comes at the cost of large performance overheads. We then study how exploiting standard optimization techniques from the MPC literature can be used to reduce this gap. In particular, we show that "packed secret sharing" based on a modified multiplication algorithm can speed up MPC-based masking when the order of the masking scheme increases. Eventually, we discuss the randomness requirements of masked implementations. For this purpose, we first show with information theoretic arguments that the security guarantees of masking are only preserved if this randomness is uniform, and analyze the consequences of a deviation from this requirement. We then conclude the paper by including the cost of randomness generation in our performance evaluations. These results should help actual designers to choose a masking scheme based on security and performance constraints.

1 Introduction

Masking is one of the most investigated countermeasures against side-channel attacks. Its underlying principle is to randomize any sensitive data in a cryptographic implementation, by splitting it into d shares. Intuitively, such a process is expected to "force" the adversary to combine several leakage samples corresponding to these shares, in order to recover secret information from her measurement traces. This is an arguably more difficult task than targeting single samples separately because (1) more "points of interests" (i.e. more dimensions in the leakage distribution) may have to be identified and exploited concurrently (e.g. if the shares are processed sequentially), hence increasing the time complexity of the attacks accordingly; (2) if the masking scheme is carefully implemented (e.g. avoiding the glitch issue described in [19]), higher-order moments of this leakage distribution will have to be estimated. It implies an increase of the attacks data complexity that is exponential in the number of shares (with the measurement noise variance as basis), as first hinted towards by Chari et al. in the specialized case of single-bit DPA attacks [3], then experimented by Standaert et al. in more general contexts [30], and recently shown formally by Prouff and Rivain [23], using the mutual information put forward in [29] as evaluation metric.

From a theoretical point of view, the problem of masking a cryptographic implementation has strong connections with the problem of secure MultiParty Computation (MPC). This observation was already made in 2004 by Ishai et al. [15], and intuitively corresponds to the fact that both masking and MPC aim to perform computations on shared data. Their objectives are different though, as MPC protocols usually fulfill stronger security requirements (they typically remain secure after the corruption of a number of participants, in an passive or active manner). By contrast, masking only aims at ensuring the so-called d-th order security property, i.e. that every d-tuple of intermediate values in the target implementation is independent of any sensitive variable. Since the performance overheads of masking and MPC are generally important, a natural problem is to determine the physical security advantages of MPC over masking, as well as its cost penalty. From the security point of view, an answer to this question has been put forward by Roche and Prouff in [25]. Namely, implementing MPC can lead to glitch-free implementations (in a similar sense as first described in [21]), and allows fault-tolerance if active adversaries are considered. By contrast, their performance evaluations were limited to asymptotic complexities so far.

Our contribution. In this paper, we investigate this performance gap between masking and MPC in the practically relevant case of AES implementations in an 8-bit microcontroller. We considered three different directions for this purpose.

First, we compared a number of existing schemes. Our selection was motivated by the two following criteria: (*i*) exclude "broken" proposals (i.e. with low-order weaknesses), such as the multiplicative masking in [13], the higher-order masking in [27] (broken in [4]), or Goubin and Martinelli's proposal in [14] (broken in [26]); (*ii*) exclude schemes that do not systematically generalize to higher-orders, such as the affine masking in [10, 32], the threshold implementations in [20], and several ideas from the "early" DPA literature (see [18] for a survey)[1]. This essentially leaves us with Rivain and Prouff's higher-order Boolean masking scheme from CHES 2010 [24] (next denoted as **RivP**), its optimization by Kim et al. using extension fields for the AES S-box implementation in [16] (next denoted as **KHL**), Genelle et al.'s solution based on the switching between additive and multiplicative masking [12] (next denoted as **GPQ**), and the MPC-inspired proposal by Roche and Prouff from CHES 2011 [25] (next denoted as **RocP**). We implemented these different schemes up to the 10th security order, with results illustrating a large gap between the MPC-inspired **RocP** (for which we additionally propose a slight optimization) and other masking schemes.

Motivated by this large performance gap, we then investigated a standard solution used in the MPC literature to improve performances, namely "packed secret sharing" [9]. In particular, we evaluate the extent to which the techniques proposed by Damgård et al. in [5] can be used to enhance the performances of shared AES implementations, and how this performance gain depends on the order d. Intuitively, the idea of packed secret sharing is to "hide" several

[1] We also excluded the recently proposed "inner product" masking scheme from [1], although it is certainly an interesting scope for further investigation.

secrets (e.g. key bytes) in a high-degree polynomial, which leads to more efficient computations if operations on these secrets can be performed in parallel. We show that such a technique is indeed useful for protecting the AES S-boxes, and exhibit the linear amortized complexity that it allows. Yet, we also show that this amortized complexity only becomes beneficial for quite large orders.

Eventually, we tackled a usually neglected problem in the literature on masking, namely the randomness requirements. First, we briefly discuss the impact of slight defaults in the Random Number Generator (RNG) used to produce fresh shares. In particular, we provide an information theoretic evaluation of the cases where (*i*) the RNG has a small bias, and (*ii*) a counter was used to generate equally likely but predictable outputs. This evaluation naturally suggests that uniform randomness is a strong requirement for the security of masking (and MPC). Then, we evaluated the performances of our different masking schemes again, including the cost of (strong-enough) randomness generation.

Overall, these result allow an implementer to decide which state-of-the-art masking scheme to use and why, in function of his security goals (in terms of order of the scheme and glitch-freeness), and performance constraints.

Methodology. As clear from the previous introduction, our goal is to compare the performances of a large number of masked implementations, up to high security orders. Relying exclusively on optimized assembly language was out of reach in this context. As a result, we systematically took advantage of C language descriptions, and paid a particular attention in optimizing them in such a way that their compilation on an 8-bit device was close enough to the one of published implementations. In particular, we used the AVR-GCC compiler (with option -o2) to obtain codes for an Atmel AtMega644p 8-bit microcontroller. And for each implementation published by independent authors (e.g. in [12, 16, 24]), we made sure that our performances were comparable up to a factor two in clock cycles. For this purpose, we relied on the optimization of certain routines (e.g. for the masking of S-boxes) whenever needed. Furthermore, we systematically wrote our codes in two fashions: one unrolled version optimized for speed and one compact version without loop unrolling. As for optimization criteria, we first focused on the cycle count, and considered the 64Kb of our target device as a memory constraint to reach. In view of the larger performance differences that will be put forward between the investigated masking schemes and security orders, we believe this methodology was sufficient to support our conclusions.

2 Comparison & improvement of existing schemes

2.1 Description of selected schemes

In this first section, we aim to compare AES implementations protected with various masking schemes. For this purpose, a preliminary observation is that the AES is composed of operations MixColumns, ShiftRows, AddRoundKey and SubBytes. Since the Boolean and polynomial masking schemes on which we will focus are bytewise XOR-linear, the operations MixColumns, AddRoundKey and

ShiftRows can be executed independently on each share. As a result, we now focus on the description of SubBytes for efficient masking. This operation executes 16 nonlinear S-boxes in parallel, for which several representations exist.

For Boolean masked implementations and for the switching method **GPQ** we used the standard representation, combining an inversion in $GF(256)$ and an affine transform. This is naturally motivated by the fact that the affine transform is bitwise XOR-linear. In this case, the most difficult operation is the inversion, which is best achieved by exploiting secure multiplications for **RivP**, as described in [24], Algorithm 1. **KHL** is based on similar ideas, but exploits subfields to reduce the cost of field multiplications and the amount of randomness. By contrast, in the case of **GPQ** the switch allows moving from a Boolean masking scheme to a multiplicative-linear one, which makes the inversion easy and defers most the complexity to the switch operation. Algorithms 1 and 2 in [12] describe how to perform this change securely. The main challenge of this solution is to pay attention to the masking of the zero value in the multiplicative masking. In order to solve this issue, the authors compute the Dirac value of the secret (which can be done efficiently by computing 8 such values concurrently, as described in [11], Algorithm 4). Since these techniques are now standard in the CHES community, we refer to the original papers for the technical details.

For the polynomial masking **RocP**, we note that no implementation results have been provided so far. This solution essentially exploits core ideas from the MPC literature. In particular, it shares the sensitive values in an implementation using Shamir's trick [28], and computes on these shares securely using the results of Ben-Or, Goldwasser and Widgerson [2]. A brief summary of these techniques is provided in Appendix A for the unfamiliar reader. In this context, an important observation is that the scheme is not bitwise XOR-linear. As a result, the best S-box representation is in polynomial form, namely $\mathsf{0x63} + \mathsf{0x5}\ x^{-1} + \mathsf{0x9}\ x^{-2} + \mathsf{0xf9}\ x^{-4} + \mathsf{0x25}\ x^{-8} + \mathsf{0xf4}\ x^{-16} + x^{-32} + \mathsf{0xb5}\ x^{-64} + \mathsf{0x8f}\ x^{-128}$. Again, the most difficult part of this S-box is the inversion, which can be implemented using 4 multiplications and some squarings. Roche and Prouff describe a polynomial multiplication in [25], Algorithm 1. Since the focus of their work was on glitch-freeness, they proposed to use a $(2d+1, d)$-sharing for all operations, including linear ones (which allows separating the implementation in several independent sub-circuits). In the following, we suggest a slight modification of this proposal which essentially extends a $(d+1, d)$-sharing to a $(2d+1, d)$-sharing in a glitch-free manner (as described in Algorithm 1). This tweak will be denoted as **RocP**$\star$. It allows us to perform the linear operations with a lower degree sharing, and to divide by 2 the cost of these operations in our masked implementations.

In this algorithm, we use $m \in_R GF(256)$ to mean that m is uniformly randomly chosen in $GF(256)$. The coefficients $\lambda_i^j = \prod_{0 \leq k \leq d} \frac{x_j \oplus x_k}{x_i \oplus x_k}$ (with $k \neq i$) are the evaluations in x_j of the Lagrangian of the $(d+1, d)$-sharing. Since the points x_i's are chosen before the execution of the masked implementation, these λ_i^j can be precomputed. Interestingly, the different shares are always used one at a

Algorithm 1 Expanding of a sharing

Require: A $(d+1, d)$-sharing $(x_i, y_i)_{i=1}^{d+1}$.
Ensure: A $(2d+1, d)$-sharing $(x_i, t_i)_{i=1}^{2d+1}$.

for j from 1 to $d+1$ **do**
    $t_j = y_j$
end for
for j from $d+2$ to $2d+1$ **do**
    for i from 1 to $d+1$ **do**
        $m_i^j \in_R GF(256)$
        $tmp_i^j = y_i \otimes \lambda_i^j$
        $tmp_i^j = tmp_i^j \oplus m_i^j$
    end for
    $t_j = 0$
    for i from 1 to $d+1$ **do**
        $t_j = t_j \oplus tmp_i^j$
    end for
    for i from 1 to $d+1$ **do**
        $t_j = y_j \oplus m_i^j$
    end for
end for
return $(x_i, t_i)_{i=1}^{2d+1}$

time in Algorithm 1. Hence, just as in **RocP**, no glitches can leak information on several shares and the implementation of this algorithm can be based on separate sub-circuits. Namely, a first class of sub-circuits calculates the masked values and sends the information at the right time; a second class of sub-circuits combines the information to obtain new shares. Note finally that our implementation did not take advantage of the DFT technique proposed in [26] since for the security degrees we considered, it did not lead to significant performance gains[2].

2.2 Implementation result

We now compare the performances of the selected schemes, considering both unrolled and compact implementations. As previously mentioned, the use of unrolled codes allows reducing the execution time at the cost of increased code size. Hence, it is limited to lower security orders in our target devices. Figure 4(a)

[2] For example, to evaluate a polynomial of degree 16 in 16 points, our basic method requires 256 multiplications and 256 XOR's. For the same evaluation with the DFT solution, we have to reduce a 16-degree polynomial by a 16-degree polynomial, which requires 17 multiplications, 16 XOR's, and 1 inversion. Then two reductions of a 15-degree polynomial by an 8-degree polynomial have to be performed, each of them requiring 72 multiplications, 64 XOR's and 1 inversion. Eventually, the DFT technique corresponds to 321 multiplications, 256 XOR's and 15 inversions in this case. Since the maximum degree we will consider in our experiments is 12, and DFT-based evaluations work best with powers of 2, we believe it will not lead to significant improvements and focus on other possible optimizations in the next section.

contains the execution times of the masked AES implementation in unrolled version (up to security order 7). Figure 4(b) exhibits similar results in the compact implementation case (up to security order 10). As can be observed, this programming style has a significant influence on the cycle counts.

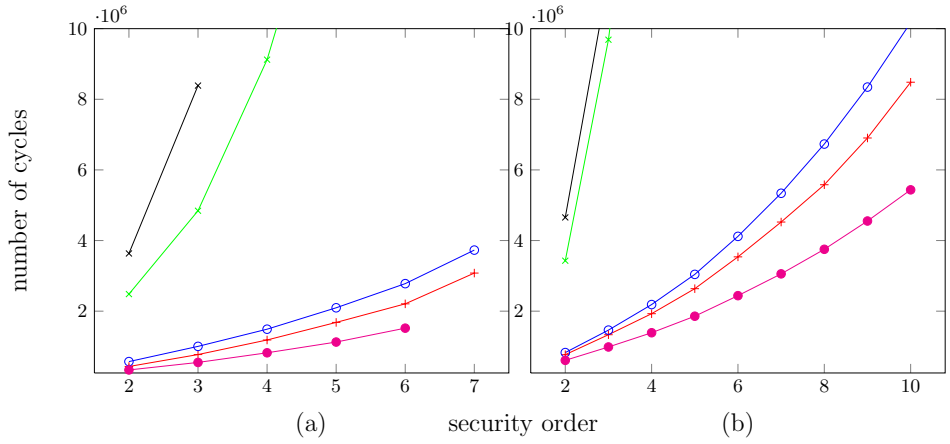

Fig. 1: Cycle counts for masked AES implementations: unrolled codes (a) and compact codes (b). The curves —+— are for **KHL**, the curves —×— are for **RocP**, the curves —○— are for **RivP**, the curves —×— are for **RocP⋆**, and the curves —●— are for **GPQ**.

These first figures clearly illustrates the significant performance gap between "standard" masking schemes and the MPC-based solution **RocP**. We also observe that our tweak for **RocP** leads to interesting gains, in particular in the case of unrolled codes (indeed, step 4 in Algorithm 1 can be performed with a single table access in this case). Eventually, the switching method **GPQ** provides the most efficient implementations, which connects with previously published results and the intuition that the AES is particularly well suited to this solution, since it alternates XOR-linear parts and multiplications. As for Boolean masking, the advantage of the subfield representation in **KHL** is also observed.

3 More efficient MPC with packed secret sharing

The previous section suggests that the polynomial masking scheme **RocP** suffers from significant performance overheads compared to **GPQ** or **RivP-KHL**. Hence, despite its interesting security features (e.g. in terms of glitch-freeness, or ability to prevent fault attacks), the gap between the security orders that can be reached with one or the other type of masking clearly benefits to the simplest solutions. Yet, the proposal by Roche and Prouff was mainly based on early results in the MPC literature. As a result, this section investigates whether some more recent optimizations could be exploited to improve the performances of MPC-based masking. In particular, we evaluate the opportunities to take advantage of

packed secret sharing. The main idea of this technique is to hide several secrets in a higher-degree polynomial, by using several initial conditions (see Appendix A). The opening is then performed by evaluating the polynomials in the locations used by the dealer. In general, such a solution is useful when there is exploitable parallelism in the algorithm to execute. In the following, we will focus on the parallelism available in the execution of the `SubBytes` transform.

3.1 Description of the packed secret sharing techniques

Intuition. The packed secret sharing technique essentially consists in hiding several secrets in the same polynomial, in order to amortize the cost of computing a function over several masked secrets in parallel. Let t be the number of secrets (e.g. corresponding to the number of S-boxes to execute in our AES case), and d the threshold number (i.e. the security order of the masking scheme). Suppose that a single masked S-box has cost of $O(d^2)$ basic field operations (and assuming that all constants hidden by the O notation are small). A naive way to execute the t S-Boxes in parallel would require cost of $O(td^2)$. By exploiting the properties of the packed secret sharing, this can be reduced to $O(t+d)^2$. As a result, for a fixed d and in a setting where $t \simeq d$ operations are executed in parallel, the complexity of the protected evaluation is increased asymptotically only by a linear factor compared to an unprotected evaluation. Notice however that this improvement in complexity is only achieved for circuits of reasonable size.

How to multiply. Using packed sharing prevents to use secure multiplications based on Ben-Or et al. [2]. Indeed, to reduce the degree of the polynomial their solution is to erase all the large monomials. This can be done if the secret is located in 0 (since the elimination of large monomials does not change the secret in this case). But packed secret sharing needs several locations for the secrets, which implies that the truncation of polynomials becomes difficult to realize, as best illustrated with the following example. Let s_1 (resp. s_2) be a secret shared by a polynomial $P_1(X)$ (resp. $P_2(X)$). $s_l = P_l(0) = \sum_i t_i^{(l)} \prod_{j \neq i} \frac{x_i}{x_j - x_i}$, for l =1 or 2 . Remark that we can write $P_l(X) = \sum_{0 \leq i \leq d} a_i^{(l)} X^i$. Ben-Or et al. et al. calculate $s_1 s_2$ by performing the product $Q(X) = P_1(X)P_2(X)$. The polynomial $Q(X)$ has a degree $2d$, but since the secret is located in 0, it can be truncated by securely erasing all monomials larger than d. Note that this secure erasure process requires to combine different shares. Hence the information exchanged in this step needs to be masked in order to maintain the security order. Let this truncated polynomial be denoted as $Q|_d(X)$. Then we have $Q(0) = Q|_d(0)$. All other evaluations can be affected by the erasure of the largest monomials.

A natural solution to avoid this problem is to rely on a different multiplication algorithm. For example, we can use the proposal by Damgård et al. [5], described in Algorithm 2. In brief, this multiplication masks the result with a random polynomial, opens the result and finally removes the random polynomial, for a

Algorithm 2 Polynomial opening multiplication

Require: A (n, d)-sharing of y and $z : (x_i, y_i)_{i=1}^n$ and $(x_i, z_i)_{i=1}^n$.
Ensure: A (n, d)-sharing of $y \times z : (x_i, t_i)_{i=1}^n$.
1: **for** i from 1 to n **do**
2: $r_i \in_R GF(256)$
3: **end for**
4: Use Alg. 1 on $(x_i, r_i)_{i=1}^n$,$(x_i, y_i)_{i=1}^n$ and $(x_i, z_i)_{i=1}^n$
5: **for** i from 1 to $2n - 1$ **do**
6: $p_i = y_i \otimes z_i \oplus r_i$
7: **end for**
8: $(s_1, \ldots, s_t) = Open((x_1, p_1), \ldots, (x_{2n-1}, p_{2n-1}))$
9: $(t_1, \ldots, t_n) = Share(s_1, \ldots, s_t)$
10: **for** i from 1 to n **do**
11: $t_i = t_i \oplus r_i$
12: **end for**
13: **return** $(x_i, y_i)_{i=1}^n$

complexity in $O(d^2)$. More precisely, let $n = d+t$ and *Open* / *Share* refer to the operations that allow recovering several shared secrets and to distribute them among participants (as explained in Appendix A). The first step in Algorithm 2 is the same as in Ben-or et al, i.e. we simply calculate $Q(X) = P_1(X)P_2(X)$. Then the reduction step is different. The polynomial $Q(X)$ is masked by a random polynomial $R(X)$, which is done by adding the shares r_i's in step 6. Afterwards, the polynomial $Q + R(X)$ is evaluated in the positions where the secrets are located. Let $\{v_k\}_{k=1}^n$ be the set of the locations for the different secrets. Remark that to determine $R(v_k)$, one has to estimate n points of $R(X)$, which allows maintaining the d-th order security. Eventually, the k secrets are shared in a new polynomial $Q'(X)$ (in step 9), that is of degree n and corresponds to a sharing of $Q + R(v_k)$. Hence, it just remains to remove the random polynomial to obtain a sharing of $Q(v_k)$ of degree at most n, which is done in step 11 of the algorithm.

Squaring issues. The problem of moving the position of shares between participants when squaring (described in Appendix A) also becomes more critical when exploiting packed secret sharing. That is, we now have to face the fact that multiple secrets are hidden in several positions, which can also move. Since the secrets need to be located at the same position to be combined, we cannot use a stable set of secrets like proposed for sharing in [25]. As a result, we avoid the squaring problem directly by implementing them with secure multiplications. One consequence of this choice is that it is also interesting to use a modified addition chain for the inversion, in order to minimize both the number of multiplications and squarings. We used the one described in [6] for this purpose, which requires 11 secure multiplications (including the squarings).

Full AES. We face one more problem when extending the packing towards the full AES. Namely the operations MixColumns and ShiftRows need to move secrets that are hidden in the same polynomial. Hence, and in order to benefit both from

the performance gains of packed secret sharing during the execution of the S-boxes and from the linear parts, we decided to switch to single secret polynomials after each execution of `SubBytes`. To switch from t polynomials of degree $d+1$ (with secrets located in position 0) to a single polynomial of degree $d+t$ (with secrets located at positions v_k), we first move the positions of all the secrets from 0 to v_k (secret per secret). Next, we multiply the resulting polynomial by another polynomial, that cancels at positions $v_{j\neq k}$ and equals 1 in v_k. Eventually, we add all polynomials together to obtain a single polynomial of degree $d+t$ and containing t secrets. For the inverse operation (i.e. moving from one polynomial to t polynomials), we first move each secret from position v_k to position 0. Then we erase all monomials of high degree in order to keep a polynomial of degree d. Eventually, we refresh the masking by adding a polynomial sharing of zero. These two operations are detailed in Appendix B, Algorithms 3 and 4.

3.2 Implementation result

We now compare **RocP⋆**, with a MPC-inspired masking exploiting Algorithm 2, for various amounts of secrets hidden per polynomial. In order to evaluate the extent to which the packed secret sharing is exclusively useful for the execution of the AES S-boxes or if the switching between single-secret polynomials and packed secret polynomials is an efficient solution, we provide performance results both for 16 inversions and for the full AES. Besides, and in order to reflect the impact of high security orders, we focused on the compact versions of our codes (the impact of unrolling is very similar to the one in the previous section).

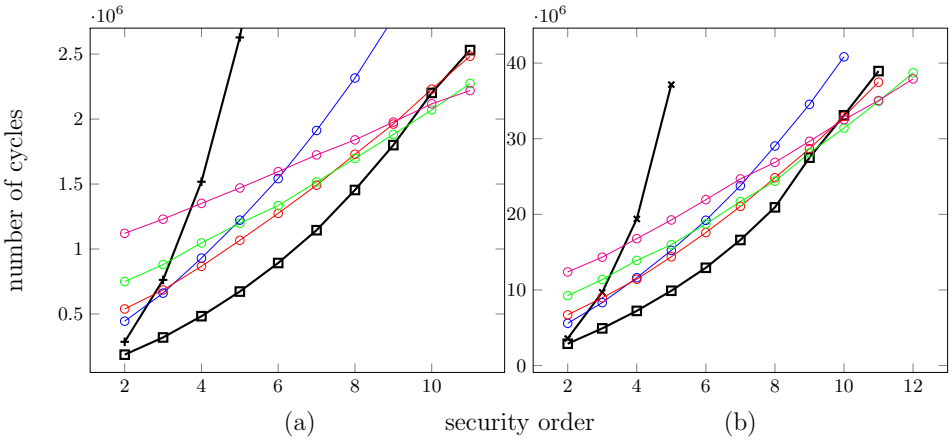

Fig. 2: Cycle counts for MPC-based masking (compact codes): 16 inversions (a) and full AES (b). The curves ⟶ are for **RocP⋆**, the packed secret sharing curves are represented by ⟶, ⟶, ⟶ and ⟶ for respectively 2, 4, 8 and 16 secrets. The curves ⟶ are for the multiplication of Algorithm 2 with a single secret per polynomial.

The results in Figure 2 first exhibit that a change of multiplication algorithm is anyway beneficial to performances in our implementation context. In particular, it is interesting to notice that the different asymptotic complexities for the multiplication in **RocP**$\star$ (cubic in the security order) and the one of Algorithm 2 (quadratic in the security order) are nicely reflected in the plots. The impact of packed secret sharing is also put forward. In particular, we can observe the (expected) quasi-linear complexity of these schemes. Interestingly, the results for the 16 inversions and for the full AES do not strongly deviate, hence suggesting that the iterated execution of Algorithms 3 and 4 does not harm performances to the point where it would become useless. Eventually, the security orders for which the quasi-linear complexity of packed secret sharing materializes remain quite high ($d = 10$ for the full AES), hence suggesting that hiding a single secret per polynomial remains the best approach in most practical settings.

4 Randomness requirements and impact on performances

Before to conclude, we would like to briefly investigate the issue of random number generation that is usually neglected in the evaluation of masking schemes. This is an important issue since the amount of randomness required to mask each non-linear operation within the AES while maintaining a security of order d is again quadratic in d. For this purpose, we will start by providing some information theoretic intuition regarding why strong randomness is indeed needed. In particular, we will show that this randomness has to be uniform, and that different deviations from this requirement imply weaknesses appearing for low and high measurement noise levels, respectively. Then, we will re-evaluate the performances of the best masking schemes we analyzed in this paper, considering a realistic performance penalty for the generation of each random byte.

4.1 How good must the randomness be?

In order to answer this question, we first repeated exactly the information theoretic analysis described in [29] and applied to the masking countermeasure in [30]. It leads to the information theoretic curves for the unprotected S-box and the 1st-order masked one in Figure 3. As detailed in these previous works, such information theoretic curves provide an evaluation of the worst-case security level of a countermeasure (i.e. the security level in front of an adversary with a perfect knowledge of the leakage distribution). In the case of masking, the order of the countermeasure is reflected in the slope of the curve (i.e. one for the unprotected S-box, 2 for the 1st-order masked one). We then considered two additional scenarios where the randomness was not as perfect as expected.

In the first place, we considered the random number to be predictable. In particular, we took the simple case where the 16 S-boxes were masked with a counter. In this context, an observation already made in [31] in the context of the shuffling countermeasure is that an adversary will be able to target the 16 masks jointly. That is, since there are only 256 possible start values, she can

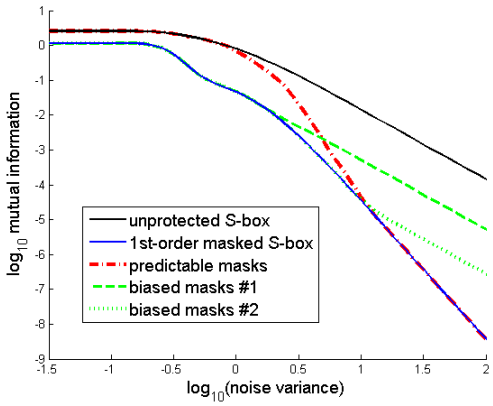

Fig. 3: Information theoretic evaluation of masking schemes.

evaluate the likelihood of the 256 sequences of 16 mask leakages, and use them in her template attack. Just as for the case of shuffling, the impact of such an imperfection of the randomness is that for low noise levels, all the masks will be recovered with probability one, as illustrated in Figure 3. As a complement, we also considered the case where the randomness was slightly biased in the same figure. Interestingly, it is well known that such biases directly create a lower-order weakness (e.g. like the "zero problem" in multiplicative masking [13]). But in fact, depending on the strength of the bias, this first-order weakness may or not be the best way to attack. That is, as illustrated in the figure, a small first-order weakness will only dominate at the noise level for which its bias is significant in front of the second-order information that is anyway available. The combination of these observations naturally suggests that both for low and high noise levels, exploiting biased or predictable randomness is not an option for masking.

4.2 Implementation result

Since strong randomness is anyway required for masking to lead to its expected security improvements, we finally repeated our performance evaluations assuming a reasonable cost for producing each random byte. Namely, we considered 10 cycles for each of these generations (excluding the memory accesses), which corresponds both to the typical quantity that we found for security chips of the same manufacturer as our target device, and to the execution of two AES rounds for producing 16 bytes of pseudorandomness. Besides, and in order to optimize the randomness requirements, we also modified the addition chain for the inversion in order to minimize this additional criteria, as proposed in [6]. We then compared the schemes of Section 2 again, namely **RivP**, **KHL**, **GPQ** and **RocP**$\star$, as well as the MPC-based scheme using the multiplication of Algorithm 2 using a single secret per polynomial, and the best packed sharing scheme from the previous section (i.e. the most efficient solution for each security degree),

considering compact codes. As illustrated in Figure 4, the cost of the random generation shifts the performance curves. But since all algorithms have a cost in randomness that is quadratic in the order of the masking scheme, this shift does not contradict the previous observations. One can just observe that the order for which packed secret sharing becomes a useful alternative is delayed by one. For the rest, the gap between **RivP**, **KHL**, **GPQ** and MPC-based masking remains large, but has been significantly reduced thanks to our optimizations.

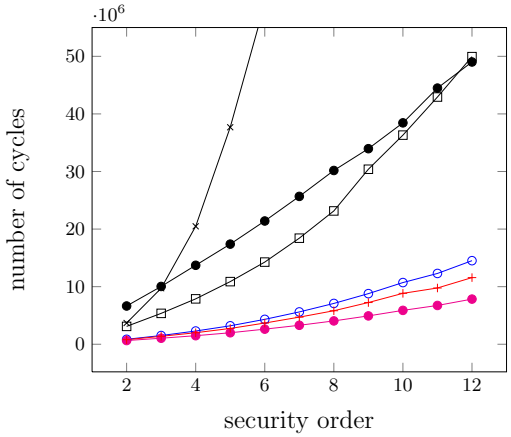

Fig. 4: Cycle counts for masked AES implementations with time to generate random. The curve —+— is for **KHL**, the curve —×— is for **RocP⋆**, the curve —○— is for **RivP**, the curve —●— is for **GPQ**, the curve —□— is for the multiplication of Algorithm 2 with a single secret per polynomial and the curve —●— is for the best packed secret sharing.

5 Conclusions

The choice of a masking scheme to protect AES implementations is a delicate tradeoff between security and performances. In this paper, we provided a careful comparison of different state-of-the-art proposals for this purpose, together with a cautionary note regaring the importance of relying on strong randomness in this context. We hope that it will help actual designers in choosing the solution that best fits their security and performance constraints. Interestingly, and despite the intuitive connection between these problems, our results show that specialized masking schemes that only guarantee higher-order side-channel security have significantly better performances than general MPC-inspired solutions. Yet, the latter ones provide interesting security features, e.g. the glitch-freeness previously discussed by Roche and Prouff, or the ability to prevent fault attacks (i.e. to resist active adversaries). Quite naturally, MPC also benefits from a huge literature, and more optimization efforts could certainly be considered to further reduce the gap between these two problems. The use of somewhat homomorphic encryption

taking advantage of a preprocessing phase to reduce asymptotic complexities can be mentioned as an example [8], in particular since it has been shown to provide efficient implementations of the AES [7]. Also, the parallelism we exploit in this work was internal to the AES (i.e. based on its 16 S-boxes). This is natural since embedded applications usually encrypt only a small amount of plaintexts. But in other scenarios where many plaintexts have to be encrypted concurrently, it would be possible to take advantage of this additional (extrernal) parallelism.

Acknowledgements. Work funded in parts by the European Commission through the ERC project 280141 (acronym CRASH) and the European ISEC action grant HOME/2010/ISEC/AG/INT-011 B-CCENTRE project. F.-X. Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

References

1. Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In Wang and Sako [33], pages 758–775.
2. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *STOC*, pages 1–10. ACM, 1988.
3. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
4. Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.
5. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.
6. Ivan Damgård and Marcel Keller. Secure multiparty AES (full paper). *IACR Cryptology ePrint Archive*, 2009:614, 2009.
7. Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 241–263. Springer, 2012.
8. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
9. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 699–710. ACM, 1992.
10. Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

11. Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Montgomery's trick and fast implementation of masked AES. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 2011.
12. Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Preneel and Takagi [22], pages 240–255.
13. Jovan Dj. Golic and Christophe Tymen. Multiplicative masking and power analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.
14. Louis Goubin and Ange Martinelli. Protecting AES with Shamir's secret sharing scheme. In Preneel and Takagi [22], pages 79–94.
15. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
16. HeeSeok Kim, Seokhie Hong, and Jongin Lim. A fast and provably secure higher-order masking of AES S-box. In Preneel and Takagi [22], pages 95–107.
17. Chung Laung Liu. Introduction to combinatorial mathematics, 1968.
18. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
19. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
20. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
21. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
22. Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.
23. Emmanuel Prouff and Matthieu Rivain. Masking against side channel attacks: a formal security proof. to appear in the proceedings of Eurocrypt 2013.
24. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
25. Thomas Roche and Emmanuel Prouff. Higher-order glitches free implementation of the AES using secure multi-party computation protocols - extended version. *J. Cryptographic Engineering*, 2(2):111–127, 2012.
26. Thomas Roche, Emmanuel Prouff, and Jean-Sébastien Coron. On the use of Shamir's secret sharing against side-channel analysis. to appear in the proceedings of Cardis 2012.
27. Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
28. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

29. 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*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
30. François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.
31. Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Wang and Sako [33], pages 740–757.
32. Manfred von Willich. A technique with an information-theoretic basis for protecting secret data from differential power attacks. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 44–62. Springer, 2001.
33. Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.

A Shamir's secret sharing & BGW

The idea of sharing a secret between several persons is a problem proposed by Liu in 1968 [17]. 11 years after, Shamir described a way to solve it [28]. His solution essentially exploits the fact that Lagrangian interpolation allows to recover the polynomial of lowest degree passing through several points. As a result, to share a secret s the person who knows the secret (usually referred to as the dealer) chooses a random polynomial P, such that $P(0) = s$ (usually referred to as the initial condition). Let d be the degree of P. Then the dealer distributes the evaluations of P at different points (called the shares) to the e participants, and labels the i-th evaluation in $x_i \neq 0$ with y_i. This step is usually called the sharing. In this paper we will denote this operation by $(y_1, \ldots, y_e) = Share(s)$. Now if $f > d$ participants want to discover the secret, they use the interpolation method to find P, and then evaluate P in 0: this step is called the opening. In this paper we will denote this operation by $s = Open(y_1, \ldots, y_f)$. By contrast if $f \leq d$, then P cannot be recovered since not enough information is available. In practice, the interpolation can be done using the Lagrangian method. In that case, the participants build $Q(X) = \sum_{0 \leq i \leq f} y_i \prod_{j \neq i} \frac{X - x_i}{x_j - x_i}$. Since this polynomial has degree at most f and verifies $Q(x_i) = y_i \ \forall i \in \{0, \ldots, f\}$, we directly have that if $f > d$, then $Q = P$ according to the fundamental theorem of algebra.

The original MPC techniques in [2] essentially aim at computing on secrets shared according to Shamir's trick. In this context, it is easy to see that the addition of two secrets can be done directly, by simply performing the addition on each pair of shares. By contrast, multiplying two shared secrets is more difficult,

since the multiplication of two polynomials of degree d in a field gives rise to a polynomial of degree $2d$. As a result, and in order for the degrees of the polynomials to remain low enough so that MPC remains efficient, it is necessary to reduce this polynomial securely. The solution proposed by Ben-Or, Goldwasser and Wigderson is to use sharings with $t > 2d$, perform the multiplications locally, and then securely delete in all the monomials of degree higher than d [2].

Note that when using polynomial masking (e.g. based on Shamir's secret sharing), the square function is also a bit more difficult to implement than in the Boolean case. Indeed, let $s = P(0) = \sum t_i \prod_{j \neq i} \frac{x_i}{x_j - x_i}$. To calculate the square of s, the participants have to compute $s^2 = \sum y_i^2 \prod_{j \neq i} \frac{x_i^2}{x_j^2 - x_i^2}$. But without special care, this operation moves the position of the shares between participants (while the execution of linear operations can only be performed if the shares are located at the same place). In [25], the authors propose to use a set S of location points that are stable by Frobenius application to avoid this problem. That is, for our case we select points such that $x^2 = y$ and then let the participants exchange shares (which is possible in the context of masking where all participants are on the same chip and assumed to be honest - but not in the general MPC case).

B Switch packed secret single secret

We describe how to switch from a single polynomial masking to a packed secret sharing (and vice versa) in Algorithms 3 and 4. Note that step 5 in Algorithm 1 allows to obtain a (n, d)-sharing from a $(d + 1, d)$-sharing. A_i^k is the evaluation on x_i of the polynomial $A_k(X) = \prod_{i \neq k} \frac{X - v_i}{v_k - v_i}$. It easy to check that this polynomial verifies the condition of annihilation in v_i for $i \neq k$. Eventually, $m_i^k = \prod_{j \neq i} \frac{x_j}{v_k - x_j}$ allows to move the location of a secret from 0 to v_k, since $m_i^k = \prod_{j \neq i} \frac{x_j}{x_i - x_f} (\prod_{j \neq i} \frac{v_k - x_j}{x_i - x_j})^{-1}$. As a result, the reconstruction with $\prod_{j \neq i} \frac{v_k - x_j}{x_i - x_j}$ will give the same secret as the evaluation in zero of the original polynomial. Similarly, $d_i^k = \prod_{j \neq i} \frac{v_k - x_j}{x_j}$ allows to move the location of the secret for v_k to zero.

Algorithm 3 Switch from t single-secret polys to 1 packed secret poly

Require: t $(d+1,d)$-sharings of $s^k : (x_i, y_i^k)_{i=1}^{d+1}$.
Ensure: A (n,d)-sharing of $\{s^k\}_{k=1}^t : (x_i, t_i)_{i=1}^n$.

for k from 1 to t **do**
  for i from 1 to d **do**
    $y_i^k = y_i^k \otimes m_i^k$
  end for
  Use a modified Alg. 1 on $(x_i, y_i^k)_{i=1}^{d+1}$
  for i from 1 to d **do**
    $y_i^k = y_i^k \otimes A_i^k$
  end for
end for
for k from 1 to t **do**
  for i from 1 to $t+d$ **do**
    $t_i = t_i \oplus y_i^k$
  end for
end for
return $(x_i, t_i)_{i=1}^n$

Algorithm 4 Switch from 1 packed secret poly to t single-secret polys

Require: A (n,d)-sharing of $\{s^k\}_{k=1}^t : (x_i, t_i)_{i=1}^n$.
Ensure: t $(d+1,d)$-sharings of $s^k : \{(x_i, y_i^k)_{i=1}^{d+1}\}_{k=1}^t$.

for i from 1 to t **do**
  for k from 1 to $d+t$ **do**
    $y_i^k = t_i \otimes d_i^k$
  end for
  for i from 1 to d **do**
    $r_i \in_R GF(256)$
  end for
  Use Alg. 1 on $(x_i, r_i)_{i=1}^{d+1}$
  for i from 1 to $d+t$ **do**
    $y_i^k = y_i^k \oplus r_i$
  end for
  $res = Open((x_0, y_0^k), \ldots, (x_{t+d}, y_{t+d}^k)$
  $(t_1, \ldots, t_{d+1}) = Share(0)$
  for k from 1 to $d+1$ **do**
    $y_i^k = r_i \oplus res \oplus t_i$
  end for
end for
return $\{(x_i, y_i^k)_{i=1}^{d+1}\}_{k=1}^t$