HAL
open science

Bitslice Masking and Improved Shuffling: How and When to Mix Them in Software?

Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos

To cite this version:

Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos. Bitslice Masking and Improved Shuffling: How and When to Mix Them in Software?. IACR Transactions on Cryptographic Hardware and Embedded Systems, In press, pp.140-165. 10.46586/tches.v2022.i2.140-165 . hal-03431477

HAL Id: hal-03431477

https://hal.science/hal-03431477v1

Submitted on 12 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Bitslice Masking and Improved Shuffling:
How and When to Mix Them in Software?

Melissa Azouaoui[1,2], Olivier Bronchain[1], Vincent Grosso[3]
Kostas Papagiannopoulos[2,4], François-Xavier Standaert[1]

[1] ICTEAM Institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium.
[2] NXP Semiconductors, Hamburg, Germany.
[3] CNRS/Laboratoire Hubert Curien, Université de Lyon, France.
[4] University of Amsterdam, Security by Design group.

Abstract. We revisit the popular adage that side-channel countermeasures must be combined to be efficient, and study its application to bitslice masking and shuffling. Our contributions are threefold. First, we improve this combination: by shuffling the shares of a masked implementation rather than its tuples, we can amplify the impact of the shuffling exponentially in the number of shares, while this impact was independent of the masking security order in previous works. Second, we evaluate the masking and shuffling combination's performance vs. security tradeoff under sufficient noise conditions: we show that the best approach is to mask first (i.e., fill the registers with as many shares as possible) and shuffle the independent operations that remain. Third, we discuss the challenges for implementing masking and shuffling under low noise conditions: we recall that such algorithmic countermeasures cannot be implemented securely without a minimum level of physical noise. We conclude that with moderate but sufficient noise, the bitslice masking + shuffling combination is relevant, and its interest increases when randomness is expensive and many independent operations are available for shuffling. When these conditions are not met, masking only is the best option. As a side result, we improve the best known attack against shuffling from Asiacrypt 2012, which we use in our concrete evaluations.

1 Introduction

Ever since the introduction of Differential Power Analysis (DPA) by Kocher et al. [25], the idea that side-channel countermeasures must be combined to be effective has become a mantra. For example one general conclusion of the DPA book is that "*implementing a combination of several cheap countermeasures typically leads to a much better protection than one expensive countermeasure*" [27]. By "much better protection", one implicitly means that the complexity of an attack against a combination of countermeasures should be significantly higher than the sum of the complexities to attack each countermeasure separately.

Following this intuition, any pair of countermeasures could potentially be combined, raising the question whether they indeed lead to concrete benefits in terms of security vs. performance tradeoff. In this respect, it was also put

forward that the most promising countermeasures to combine are the ones that provide complementary improvements. One popular example of complementary countermeasures is hiding and masking [27]. Hiding aims at reducing the attack's Signal-to-Noise Ratio (SNR) [26], by increasing the measurement noise or reducing the side-channel signal, as for example happening when shuffling an implementation [23]. Masking aims at reducing the data-dependencies of the leakages by randomizing the intermediate computations of a cryptographic implementation thanks to secret sharing, so that side-channel analysis becomes hard [14]. The statistical confirmation that this combination is sound was then put forward in an important paper by Rivain et al. [34]. Assuming a shuffled execution of η independent operations (e.g., S-boxes), each of them masked with d shares, they showed that for a sufficient noise variance σ^2 in the leakages, the complexity to attack the corresponding implementation grows in $\mathcal{O}\big(\eta \cdot (\sigma^2)^d\big)$.

Based on this state-of-the-art, the main contribution of this paper is to improve the security, the applicability and the evaluation of this important combination of countermeasures, by following three complementary tracks:

1. In terms of security under a sufficient noise regime, the proposal of Rivain et al. increases the complexity of attacks against masked implementations by a factor η.[5] Given that shuffling can be viewed as noise emulation and masking as noise amplification, a natural question is whether one could amplify the noise emulated by the shuffled operations thanks to masking and improve this gain to a factor η^d? We answer the question positively and describe an efficient solution for this purpose, with a systematic investigation of the options to combine masking and shuffling. At high level, this improvement is obtained by shuffling noisy shares rather than noisy tuples of shares.
2. In terms of applicability, several recent advances in the analysis and design of secure and efficient software masking exploit the bitslicing concept [20,6,5]. The latter has not been studied by Rivain et al. and raises new design questions. Namely, say you want to securely implement 128 independent AND gates in a 32-bit (ARM Cortex) device. What is the best combination of bistlice masking and shuffling given that both of them "utilize" the independent operations differently? We show that the good strategy is to mask first and shuffle next. In other words, fill the 32-bit bus with shares and shuffle what remains (i.e., four 32-bit operations in this example).
3. Eventually, the security evaluations of Rivain et al. rely on simple (correlation-based) side-channel attacks [33]. This led them to the conclusion that shuffling remains useful even in low-noise regimes. Yet, recent results showed that such evaluations can significantly overestimate the worst-case security level of an implementation [10]. We therefore conclude the paper with an experimental evaluation of the "masking + shuffling" combination in an ARM Cortex-M3 and we show that profiled multivariate attacks can cancel the impact of the shuffling in this case (while masking still offers a mild security

[5] Precisely, they have this factor for non-linear operations while having better results for linear operations – see the discussion in section 4 for the details.

improvement).[6] The latter can be explained by the quite leaky memory accesses that a shuffled implementation uses intensively, and raises the question of how to limit their leakage as an interesting scope for further research.

We combine these results with a technical/consolidating contribution. Namely, we describe an improvement of the multivariate attacks against shuffled implementations proposed in [39] that we use in our concrete security evaluations.

Cautionary notes. As for any side-channel countermeasure, the security gains that can be expected when combining masking and shuffling are not unconditional. Our third contribution in particular highlights the need of physical noise to make such a combination effective. So the question we tackle is not "should I combine masking and shuffling" but "when should I do it". We therefore propose a systematic evaluation that includes all the parameters influcencing its answer (e.g., the physical noise level, but also the cost of the randomness and the amount of parallelism that can be leveraged). It allows our conclusions to apply to a wide range of algorithms and to identify the ones for which the masking + shuffling combination will be the most interesting. We also note that our analyzes are based on a bitslice parameter which, when set to its minimum value, models a non-bitslice implementation. It allows us to observe that exploiting bitslicing is always beneficial, which justifies the bitslice focus of our title.

Related works. Somewhat suprisingly, and to the best of our knowledge, there are not many papers focused on combining countermeasures (which provides an additional motivation for such studies). As far as the masking + shuffling combination is concerned, the work of Bruneau et al. studies the (different) context of masking based on shuffled tables recomputations [12], which builds on the observation that recomputed tables are attractive targets for side-channel analysis [37]. The work of Patranabis et al. implements a (hardware) combination of masking and shuffling in a Rivain et al. fashion (so their shuffling is not amplified by masking as we propose) [30]. More specifically related to shuffling, another work of Patranabis et al. shows the interest to shuffle over larger sets of operations [31], which is in line with our conclusions regarding when shuffling gains interest. More general studies like [26,22] evaluate other (mostly hardware) combinations of countermeasures. Finally, a recent paper of Coron et al. initiates a theoretical analysis of shuffling in the probing model [16]. Their result are still asymptotic but provide an alternative view on the shuffling countermeasure and raise the question whether our information theoretic evaluations and the more optimistic conclusions they lead to could be further formalized.

2 Background

In this section, we introduce the notations used in the paper, the information theoretic tools needed for our evaluations and the two side-channel countermeasures we investigate (namely masking and shuffling), together with a discussion of their quantitative (information theoretic) impact on the leakages.

[6] Similar observations have been made for shuffled table recomputations [12].

2.1 Notations

Random variables are denoted with capital letters Y and their realizations with lower cases y. We use the notation $\boldsymbol{y}$ for the vector of inputs that are sent to the independent (possibly shuffled) operations. Concretely, $\boldsymbol{y}$ is a vector of size η and each element of the vector is denoted as y_i. When additionally masking the implementation, we use the notation $\boldsymbol{y}^j$ for the j-th share of the vector $\boldsymbol{y}$, such that the element-wise addition $\sum_{j=0}^{d-1} \boldsymbol{y}^j = \boldsymbol{y}$. Operations on vectors are always element-wise. We further denote y_i^j as the j-th share of element i such that $\sum_{j=0}^{d-1} y_i^j = y_i$. Without loss of generality, we assume that the leakage of a vector is a vector and the leakage of a variable is a variable. The leakage is always denoted with l with the target random variable in superscript. For example:

- $\boldsymbol{l^{\theta}}$ is the leakage (vector) on the full permutation $\boldsymbol{\theta}$ used for shuffling.
- l^{θ_c} is the leakage on the permutation at index c.
- $l^{y_i^j}$ is the leakage on the j-th share of the i-th element of $\boldsymbol{y}$.
- $l^{y^j_{\boldsymbol{\theta}_c}}$ is the leakage obtained when accessing the j-th share of the element indexed at cycle c when shuffling with the permutation $\boldsymbol{\theta}$.

2.2 Information theoretic metrics

Next, we introduce *Information Theoretic* (IT) metrics that can be used to evaluate the effectiveness of side-channel countermeasures (and, as we will see, of adversarial strategies as well). The rationale behind this choice is that the number of traces N required to perform a (worst-case) statistical attack against a leaking implementation is inversely proportional to the *Mutual Information* $\mathsf{MI}(\boldsymbol{Y};\boldsymbol{L})$ between the secret random vector $\boldsymbol{Y}$ and the leakage $\boldsymbol{L}$ [19]:

$$N \geq \frac{cst}{\mathsf{MI}(\boldsymbol{Y};\boldsymbol{L})} \cdot \tag{1}$$

Since the cardinality of $\boldsymbol{Y}$ is generally too large to be exhausted, $\mathsf{MI}(\boldsymbol{Y};\boldsymbol{L})$ is usually computed for each element of the vector independently, in a divide-and-conquer manner. In this case, one focuses on the complexity to recover one such element which is worth:

$$N_{dc} \geq \frac{cst}{\mathsf{MI}(Y_i;\boldsymbol{L})} \cdot \tag{2}$$

The relation is given for a small constant cst (that depends on the entropy of Y_i, denoted $\mathsf{H}(Y_i)$, and the target success rate of the attack [17]). In the context of attacks against implementations where the Y_i variables and their leakages are independent, the divide-and-conquer approach does not imply information losses [21]. As will be discussed next, the situation is different when shuffling, since it creates dependencies between the leakages of different Y_i variables.

In general, computing $\mathsf{MI}(\boldsymbol{Y};\boldsymbol{L})$ requires the knowledge of the (true) Probability Density Function (PDF) of the leakage conditioned on $\boldsymbol{y}$, that we denote as

$\mathsf{f}(\boldsymbol{L}=\boldsymbol{l}|\boldsymbol{Y}=\boldsymbol{y})$. Such a knowledge is only available in simulated evaluation contexts, where the leakage function is defined by the evaluator. In this paper, and unless mentioned otherwise, we will assume that the leakages follow a (possibly multivariate) Gaussian distribution with noise covariance Σ^2 so that:

$$\mathsf{f}(\boldsymbol{l}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{l}|\mu_{\boldsymbol{y}}, \Sigma^2). \tag{3}$$

Thanks to this PDF, the conditional probability of a sensitive variable $\boldsymbol{Y}$ given the leakage, denoted as $\Pr[\boldsymbol{Y}=\boldsymbol{y}|\boldsymbol{L}=\boldsymbol{l}] := \mathsf{p}(\boldsymbol{y}|\boldsymbol{l})$, can be computed via Bayes. Assuming that $\boldsymbol{Y}$ is uniformly distributed (which is the case for the cryptographic secrets we aim to recover), it is expressed as:

$$\mathsf{p}(\boldsymbol{y}|\boldsymbol{l}) = \frac{\mathsf{f}(\boldsymbol{l}|\boldsymbol{y})}{\sum_{\boldsymbol{y}^* \in \mathcal{Y}} \mathsf{f}(\boldsymbol{l}|\boldsymbol{y}^*)}. \tag{4}$$

The MI can then be estimated by sampling, as described in [9], Equation 6:

$$\hat{\mathsf{MI}}(\boldsymbol{Y};\boldsymbol{L}) = \mathsf{H}(\boldsymbol{Y}) + \sum_{\boldsymbol{y}\in\mathcal{Y}} \mathsf{p}(\boldsymbol{y}) \cdot \sum_{i=1}^{n_t(\boldsymbol{y})} \frac{1}{n_t(\boldsymbol{y})} \cdot \log_2 \mathsf{p}(\boldsymbol{y}|\boldsymbol{l}^{\boldsymbol{y}}(i)), \tag{5}$$

where $n_t(\boldsymbol{y})$ is the number of leakage samples $\boldsymbol{l}^{\boldsymbol{y}}(i)$ against which the conditional probability distribution is "tested" in the estimation of the metric (i.e., the larger is $n_t(\boldsymbol{y})$, the more accurate is the sampled estimate of $\hat{\mathsf{MI}}(\boldsymbol{Y};\boldsymbol{L})$).

When moving from simulated evaluations to the concrete evaluation of actual leakages measured from a chip, the true leakage distribution is generally unknown. The best option of an adversary is then to approximate the leakage distribution with a model, that we next denote as $\tilde{\mathsf{m}}(\cdot|\cdot)$. This model is typically obtained by profiling the target device [15]. The amount of information that can be extracted thanks to this model is captured by the so-called *Perceived Information* (PI). As the MI, the PI can be estimated by sampling by replacing $\mathsf{p}(\cdot|\cdot)$ with $\tilde{\mathsf{m}}(\cdot|\cdot)$ in Equation 5 [9], leading to:

$$\hat{\mathsf{PI}}(\boldsymbol{Y};\boldsymbol{L}) = \mathsf{H}(\boldsymbol{Y}) + \sum_{\boldsymbol{y}\in\mathcal{Y}} \mathsf{p}(\boldsymbol{y}) \cdot \sum_{i=1}^{n_t(\boldsymbol{y})} \frac{1}{n_t(\boldsymbol{y})} \cdot \log_2 \tilde{\mathsf{m}}(\boldsymbol{y}|\boldsymbol{l}^{\boldsymbol{y}}(i)). \tag{6}$$

The PI is smaller than the MI unless a perfect model is used by the adversary. In previous use cases of the PI, this metric was mostly used to capture estimation and assumption errors caused by an imperfect leakage profiling. In the following, we additionally use it to capture suboptimal adversarial strategies designed to be computationally efficient against shuffled implementations.

2.3 Masking

General principle. Masking is a popular countermeasure against side-channel attacks [14]. It consists in representing a variable a as an encoding (tuple) which

is a vector $\boldsymbol{a}$ of d uniformly distributed elements (or shares) fulfilling $a = \sum_i a^i$. By doing so, all the sets of $d-1$ shares remain independent of the secret, which was formalised as *d-probing security* [24]. Masked implementations then aim to maintain this property through the computations. For this purpose, the generic solution is to perform linear operations share-by-share and to use masked multiplication gadgets for the non-linear operations: they allow multiplying two encodings while ensuring probing security. A prominent example is the ISW multiplication introduced by Ishai, Sahai and Wagner [24] which is recalled in Algorithm 1 where $\otimes$ denotes the multiplication in $\mathsf{GF}(2)$. It has randomness requirements that are quadratic in the number of shares.

Algorithm 1 ISW multiplication.

Input: $\boldsymbol{a}$ with $a = \sum_{i=0}^{d-1} a^i$ and $\boldsymbol{b}$ with $b = \sum_{i=0}^{d-1} b^i$.
Output: $\boldsymbol{c}$ with $c = \sum_{i=0}^{d-1} c^i$ and $c = a \otimes b$.

```
for i in [0, ..., d-1] do
    c^i ← a^i ⊗ b^i
for i in [0, ..., d-1] do
    for j in [i+1, ..., d-1] do
        r ← {0,1}
        c^i ← c^i ⊕ (r ⊕ (a^i ⊗ b^j))
        c^j ← c^j ⊕ (r ⊕ (a^j ⊗ b^i))
```

Concrete security. Duc et al. showed in [18] that probing security reduces to security in the more realistic *noisy leakage model* where the adversary has access to the noisy leakages of all the intermediate variables. The (noise and independence) conditions needed for this result to hold in practice and its connection with the previous information theoretic metrics have then been made explicit in [19], leading to the following bound for the data complexity of a side-channel attack against a masked implementation:

$$N^{msk} \geq \frac{cst}{\prod_{j=0}^{d-1} \mathsf{MI}(\boldsymbol{Y}^j; \boldsymbol{L})}. \tag{7}$$

Precisely, the exponential complexity increase is only relevant if $\mathsf{MI}(\boldsymbol{Y}^j; \boldsymbol{L})$ is small enough and the security order d is maintained as long as the leakage function is a noisy linear combination of the shares. Typical defaults that can contradict this second assumption are *glitches* in hardware [29] and *transition-based leakages* in software [4] – both of them can be prevented thanks to algorithmic tweaks. As will be clear in Section 6 these defaults are not critical in our evaluations (where the weaknesses primarily arise due to a lack of noise).

2.4 Shuffling

General principle. Shuffling is a side-channel countermeasure that leverages independent operations within a circuit (e.g., the 16 AES S-boxes). It consists in performing these operations in a random order with the goal to confuse the side-channel adversary. Algorithm 2 is an example of shuffling that applies an arbitrary function $\mathsf{op}(\cdot)$ independently to all the elements of the input vector $\boldsymbol{y}$. The first step is to generate a permutation $\boldsymbol{\theta}$ which is uniformly selected among all the permutations of the set $\{0, \ldots, |\boldsymbol{y}|-1\}$, that we denote $\boldsymbol{\Theta}$. The size of the permutation is next given by π. For now, it corresponds to η which is the number of independent operations on which we shuffle and corresponds to the size of $\boldsymbol{y}$. As will be seen next, there are also cases where $\pi > \eta$. The algorithm iterates deterministically over the elements of this permutation. On the c-th iteration, the index θ_c of the element of the input vector to process during that iteration (denoted s) is loaded. The output is finally updated such that $z_s \leftarrow \mathsf{op}(y_s)$.

Algorithm 2 Shuffled execution.

Input: $\boldsymbol{y}$ and $\mathsf{op}(\cdot)$ with $|\boldsymbol{y}| = \eta$ and $\pi = \eta$.
Output: $\boldsymbol{z}$ such that $\forall i, 0 \leq i < \pi, z_i = op(y_i)$.

```
θ ← Θ                                   ▷ Perm. generation
for c in [0, ..., π − 1] do
    s ← θ_c                             ▷ Leaks ⇝ l^{θ_c}
    z_s ← op(y_s)                       ▷ Leaks ⇝ l^{y_{θ_c}} and l^{z_{θ_c}}
```

In the context of a side-channel attack, all these operations generate leakage on the processed data. Namely, at iteration c, the c-th element in the permutation generates some leakage denoted as l^{θ_c}. We shall refer to it as the *permutation leakage*. The manipulation of the input vector also generates some leakage that we next call *data leakage*. We use the notation $l^{y_{\theta_c}}$ to represent the leakage generated when processing the θ_c-th index in the vector $\boldsymbol{y}$ during cycle c.

Concrete security. Under the assumption that the noise is sufficient to hide the permutation indexes, shuffling offers an increase of a side-channel attack's data complexity that is linear in the number of independent operations η on which the permutation is applied [23,39]. Using the same information theoretic notations as for the masking countermeasure, it gives :

$$N^{shu} \geq \frac{cst \cdot \eta}{\mathsf{MI}_u(\boldsymbol{Y}; \boldsymbol{L})}\,, \tag{8}$$

where the denominator is the MI_u on a similar un-shuffled implementation.

Computing the PDF. The optimal way to compute the leakage PDF of a shuffled implementation is given by the next equation:

$$\mathsf{f}(\boldsymbol{l}|\boldsymbol{y}) = \sum_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathsf{p}(\boldsymbol{\theta}) \cdot \mathsf{f}(\boldsymbol{l}|\boldsymbol{\theta}, \boldsymbol{y}), \tag{9}$$

where $\boldsymbol{l}$ is the concatenation of the permutation and data leakage vectors. However, summing over all the permutations rapidly turns out to be too computationally intensive (e.g., for $\pi = 16$, the number of modes of this mixture is already $\approx 2^{44.2}$). As a result, more computationally efficient approaches have been proposed, at the cost of a possible information loss.

The AC12 attack. At Asiacrypt2012, Veyrat-Charvillon et al. proposed an approach to recover $\boldsymbol{y}$ by exploiting the leakage on the permutation indexes efficiently [39], as reflected by the following equation:

$$\mathsf{AC12}(\boldsymbol{l}|y_s) = \sum_{c=0}^{\pi-1} \underbrace{\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}})}_{\text{perm. leak.}} \cdot \underbrace{\mathsf{f}(l^{y_{\theta_c}}|y_s)}_{\text{data leak.}}, \tag{10}$$

where $\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}})$ is the weight assigned to a cycle c. Its goal is to indicate the probability that the targeted value y_s is manipulated at the cycle c. They propose many solutions to derive $\mathsf{w}_{\mathsf{AC12}}$ with different time and data complexities.[7] We focus on the so-called *"direct permutation leakages"* (DPLeak), for which this weight is computed as:

$$\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}}) = \frac{\mathsf{f}(l^{\theta_c}|\theta_c = s)}{\sum_{c'=0}^{\pi-1} \mathsf{f}(l^{\theta_{c'}}|\theta_{c'} = s)}. \tag{11}$$

Putting things together, the $\mathsf{AC12}$ attack recovers the full vector $\boldsymbol{y}$ by applying Bayes on each of the elements of the vector independently, using a model:

$$\tilde{\mathsf{m}}_{\mathsf{AC12}}(\boldsymbol{y}|\boldsymbol{l}) = \prod_{s=0}^{\pi-1} \frac{\mathsf{AC12}(\boldsymbol{l}|y_s)}{\sum_{y_s^*} \mathsf{AC12}(\boldsymbol{l}|y_s^*)}. \tag{12}$$

Note that since this model is imperfect, using it leads the adversary to exploit some perceived information rather than the whole mutual information. Note also that despite this attack is not summing over all the permutations (which makes it suboptimal from the data complexity viewpoint), it is not a divide-and-conquer one since the leakages of the permutation are still exploited jointly.

3 Improving the AC12 attack strategy

We next propose an improved computationally efficient strategy in order to attack shuffled implementations, and use simulated information theoretic evaluations to demonstrate the gains it provides over the $\mathsf{AC12}$ approach.

[7] Which, for large enough noise, are equivalent from the data complexities viewpoint.

3.1 Attack specification

In principle, the attack we propose is similar to the AC12 DPLeak one. Yet, it uses a slightly different model that is expressed by the following equation:

$$\tilde{\mathsf{m}}_{\mathsf{New}}(\boldsymbol{y}|\boldsymbol{l}) = \prod_{s=0}^{\pi-1} \sum_{c=0}^{\pi-1} \underbrace{\mathsf{w}_{\mathsf{New}}(\theta_c = s|l^{\theta_c})}_{\text{perm. leak.}} \cdot \underbrace{\frac{\mathsf{f}(l^{y_{\theta_c}}|y_s)}{\sum_{y_s^*} \mathsf{f}(l^{y_{\theta_c}}|y_s^*)}}_{\text{data leak.}}, \tag{13}$$

where the first term processes the permutation leakages and the second term processes the data leakages. Our modifications compared to Equation 12 are twofold. First, the weighted sum over c is not performed on continuous PDFs anymore but on the probabilities obtained after the application of Bayes (as reflected by the right term in the above equation). This is the usual approach taken for advanced attacks such as [38]. Second, we notice that the weights in the sum estimated with Equation 11 depend on the full permutation leakage. Since each term in the sum corresponds to the leakage at cycle c, we modified the weights such that they give the probability that the index manipulated at cycle c is equal to s. Formally, this leads to:

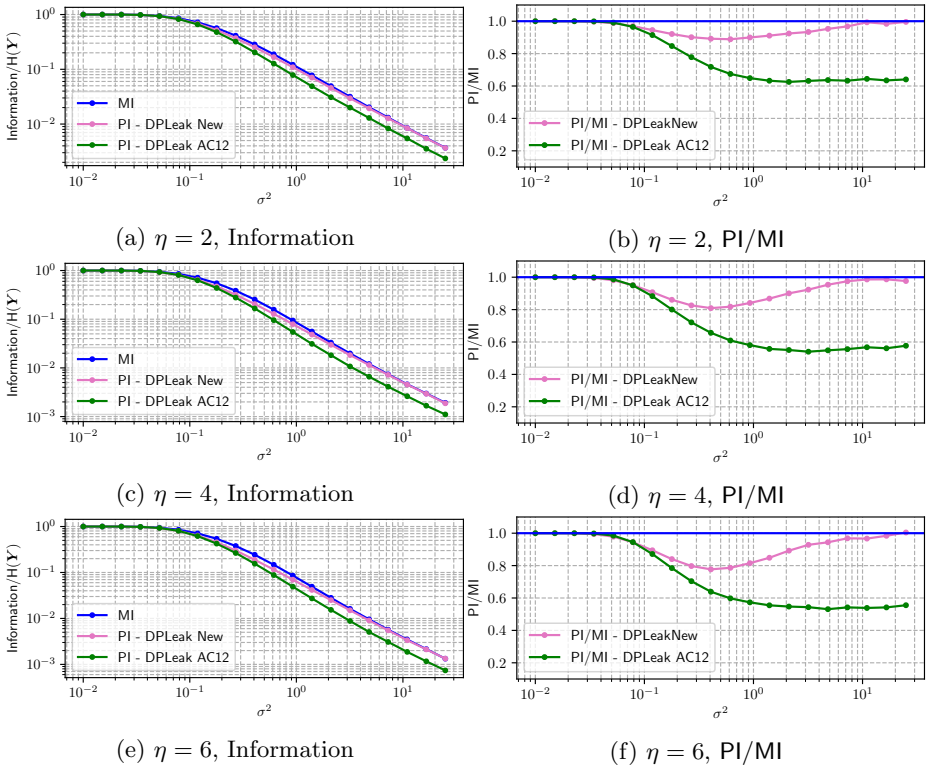
$$\mathsf{w}_{\mathsf{New}}(\theta_c = s|l^{\theta_c}) = \frac{\mathsf{f}(l^{\theta_c}|\theta_c = s)}{\sum_{s'=0}^{\pi-1} \mathsf{f}(l^{\theta_c}|\theta_c = s')}. \tag{14}$$

3.2 Models comparison

Methodology. Next, we simulate a shuffled implementation (i.e., Algorithm 2) where the leakages are distributed according to Equation 9. From these leakages and the knowledge of the true PDF, we first estimate the MI according to Equation 5 which represents the best attack possible against the implementation. We do that for small permutation sizes (since for large π values the direct computation of the MI is computationally hard). For the same implementations, we extract the PI thanks to Equation 6 for both models $\tilde{\mathsf{m}}_{\mathsf{AC12}}(\cdot|\cdot)$ (Equation 12) and $\tilde{\mathsf{m}}_{\mathsf{New}}(\cdot|\cdot)$ (Equation 13). It allows discussing which model is the best and how far it is from the optimal attack enumerating all the permutations.

Practically, the simulations take two parameters. The first one is the noise variance σ^2 in Equation 3. It represents the amount of noise that is intrinsic to the implementation. The second one is the number of independent operations on which we shuffle, which corresponds to the size η of the secret vector $\boldsymbol{y}$. As a result, we have $\mathsf{H}(\boldsymbol{Y}) = \pi \cdot \mathsf{H}(Y_i) = \eta \cdot \mathsf{H}(Y_i)$. Due to the aforementioned computational limitations, we take values $\eta \in \{2, 4, 6\}$. However, we note that the evaluation of the AC12 attack and our improvement do not suffer from such a limitation, meaning that PI could be evaluated also for a larger η.[8]

[8] The simulations performed in this work are available to the reviewers as supplementary material and will be published under an open-source license.



(a) $\eta = 2$, Information (b) $\eta = 2$, PI/MI

(c) $\eta = 4$, Information (d) $\eta = 4$, PI/MI

(e) $\eta = 6$, Information (f) $\eta = 6$, PI/MI

Fig. 1: IT analysis of shuffled implementation Algorithm 2. Normalized IT metrics are on the left. The ratios between the PIs and the MI are on the right.

Results and discussion. The results of these simulations are reported in Figure 1. On the left, we observe the resulting MI and PIs according to the noise parameter σ^2: DPLeak AC12 is the label of the model $\tilde{\mathsf{m}}_{\mathsf{AC12}}(\cdot|\cdot)$ and DPLeak New is the label of the model $\tilde{\mathsf{m}}_{\mathsf{New}}(\cdot|\cdot)$. On the right, we report the ratio between the MI and the PIs. This part of the plot is used to highlight the gap between the efficient adversaries' models and the worst-case attack. Based on these simulations, we first observe that our new model improves over the AC12 one. Indeed, the PI for the DPLeak AC12 adversary is always (sometimes significantly) lower than DPLeak New. Second, we observe that for high noise levels, our new model offers a good approximation of the MI while the AC12 one suffers from a bias.

In the rest of this paper, such small-scale information theoretic evaluations (based on the worst-case MI) will be quite systematically used in order to evaluate and compare different combinations of masking and shuffling. We therefore use this first application to discuss their main pros and cons as follows.

On the negative side, (i) they only correspond to attacks considering a representative leakage function, which is not equivalent to proving security in general: the gap between provable analyzes and worst-case attacks for shuffing is highlighted in [16] and tightening this gap is an important open problem; and (ii)

these small scale examples do not directly apply to the typical sizes of concrete implementations (e.g., $\mathsf{H}(Y_i) = 8$ and $\eta = 16$ for the AES Rijndael).

On the positive side, (i) information theoretic evaluations as we propose have quite systematically been shown to be excellent indicators of the bounds that can be obtained in masking security proofs: see for example the sequence of papers [35,36,32,18,19] for an illustration; and (ii) the same holds (up to constant factors) for the extrapolation from small variables to larger ones, and the concrete attacks we propose do not suffer from computational limitations.

Since all our observations are confirmed for a few (growing) values of d and π, we therefore believe they provide a necessary first step towards a better understanding of the masking + shuffling combination of countermeasures which, in particular, improves over the one of Rivain et al. [34], both in terms of the security levels we claim and in terms of the considered attacks' coverage.[9]

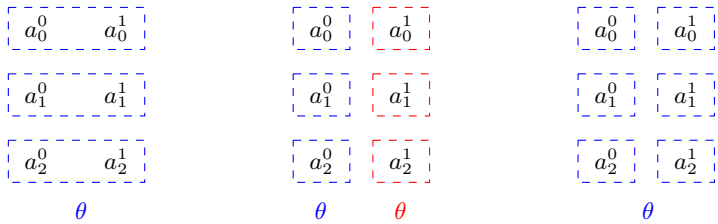
4 Systematic information theoretic analysis

As discussed in subsection 2.3, a masked d-probing secure circuit is composed of two types of of operations: linear ones can be applied share-by-share, non-linear ones require to mix the shares in a secure manner. We now analyze different approaches to combine masking and shuffling for these two types of operations. We start with paper-and-pencil intuitions to express the security gains such combinations provide in simple terms and/or to revoke some possible options. Then, we evaluate some relevant combinations with an information theoretic analysis. For now we focus on the high-noise regime, where exploiting the permutation leakages does not improve the attacks [39]. Some simulations taking into account the permutation leakages are reported in Appendix A. The case of low noise regime will be discussed based on a real-world case study in section 6.

4.1 Linear operations

Based on the notations of subsection 2.1, applying a linear operation $\mathsf{op}_{\mathsf{l}}(\cdot)$ to an encoding of the secret vector $\boldsymbol{a}$ of size η consists in applying $\mathsf{op}_{\mathsf{l}}(\cdot)$ independently to all the elements of its share vectors $\boldsymbol{a}^i$. Namely, the output encoding of $\boldsymbol{b}$ is derived such that $b^i_s = \mathsf{op}_{\mathsf{l}}(a^i_s)$ for all $0 \leq i < d$ and $0 \leq s < \eta$. The challenge when combining masking and shuffling for linear layers is to identify in what order should pairs of indexes (i, s) be used to load a^i_s. We consider three possible shuffling configurations that can be applied to a linear layer, as summarized in Figure 2 where a color denotes a permutation and a box the pair(s) (i, s) that are accessed deterministically at each cycle of that permutation.

[9] We note that the DPLeak New could possibly be further improved with analytical attacks such as [38]. However, Figure 1 shows that for high noise levels on which we will focus, DPLeak New is already close to the worst-case attack, which is in line with the observation in [3] that such analytical attacks only lead to minor improvements when aiming at recovering ephemeral secrets (like a permutation).



(a) Shuffling tuples. (b) Shuffling shares. (c) Shuffling everything.

Fig. 2: Shuffling configurations of a linear encoding for $d = 2$ and $\eta = 3$.

Shuffling tuples. A first straightforward possibility is to shuffle tuples of shares. Only the indexes s of the vectors are shuffled and the shares i of that index are accessed sequentially and deterministically. Algorithm 3 is an instantiation of such a combination and a graphical representation is given in Figure 2a.

Algorithm 3 Masking and shuffling tuples.

Input: $\boldsymbol{a}$ such that $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\mathsf{op}_{\mathsf{I}}(\cdot)$
Output: $\boldsymbol{b}$ such that $b_s = \mathsf{op}_{\mathsf{I}}(a_s) = \sum_{i=0}^{d-1} b_s^i$.

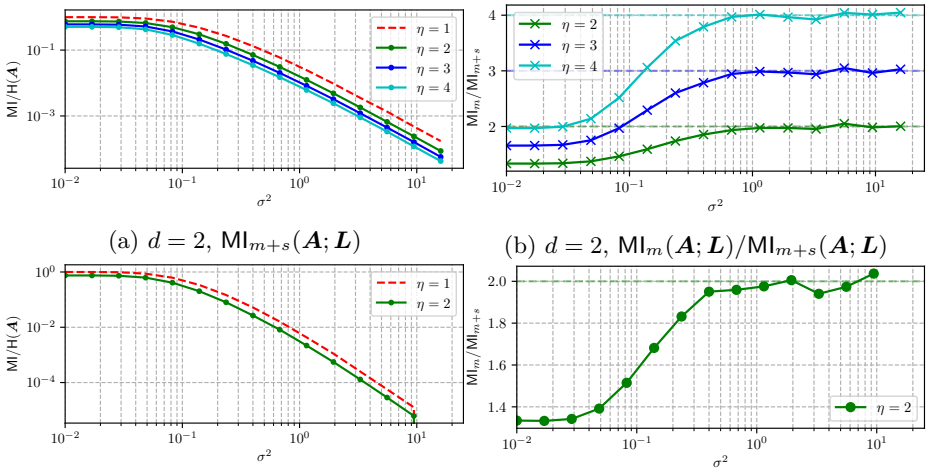
```
θ ← Θ                                   ▷ Perm. generation of size π = η
for c in [0, ..., η − 1] do
    s ← θ_c
    for i in [0, ..., d − 1] do
        b_s^i = op_I(a_s^i)
```

In terms of security, this shuffling and masking combination is similar to the case where we only shuffle an implementation from Algorithm 2. Namely, because the shares are accessed in order, at the cycle c information on all the shares $a_{\theta_c}^i$ can be combined together to get information on a_{θ_c}, without being impacted by shuffling. So similarly to a shuffled-only implementation, the security of this combination is linear in the number of operations η on which we shuffle:

$$N^{\text{tuples}} \geq \frac{cst \cdot \eta}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})} \cdot \tag{15}$$

This equation is confirmed by the IT analysis of Algorithm 3 given in Figure 3. On the left, we report the MI of a masked and shuffled implementation $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$, for various parameters d and η. On the right, we report the ratio between the MI of a masked-only implementation $\mathsf{MI}_m(\boldsymbol{A}; L)$ and $\mathsf{MI}_{m+s}(\boldsymbol{A}; L)$. Based on Equation 1, this ratio is the increase of the (worst-case) attack data complexity that shuffling and masking provide compared to masking only.

We first observe that, as expected from Equation 15, the gain is equal to the permutation size $\pi = \eta$ for sufficiently large noise variance. We additionally



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(d) $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 3: Shuffling tuples (Algorithm 3): IT analysis.

observe that this gain is independent of d, which confirms that there is no non-trivial interaction between shuffling and masking in this case. Indeed for $\eta = 2$, the gain is equal to 2 for the two and three shares implementations.

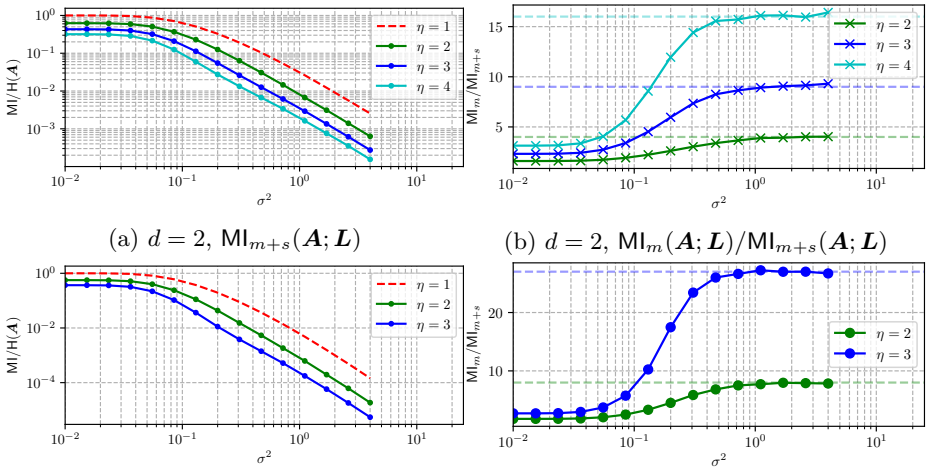
Shuffling shares. A natural option to improve the interaction between masking and shuffling is to shuffle the shares of independent variables instead of their tuples. As illustrated in Figure 2b and described formally in Algorithm 4, it consists in processing the shares sequentially and deterministically and in picking up a random permutation for each vector of shares $\boldsymbol{a}^i$. For each share index, the operations $\mathsf{op}_{\mathsf{I}}(a_s^i)$ are performed with s selected according to a fresh permutation. As a result, the permutation is always applied on independent values.

Algorithm 4 Masking and shuffling shares.

Input: $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\mathsf{op}_{\mathsf{I}}(\cdot)$
Output: $b_s = \mathsf{op}_{\mathsf{I}}(a_s) = \sum_{i=0}^{d-1} b_s^i$.

```
for i in [0, ..., d − 1] do
    θ ← Θ                                  ▷ Perm. generation of size π = η
    for c in [0, ..., η − 1] do
        s ← θ_c
        b_s^i = op_I(a_s^i)
```

Such an approach is beneficial to side-channel security since to obtain information about a secret element a_s, an adversary now has to retrieve at which cycle c the d shares a_s^i are manipulated. Without knowledge of the permutations



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (d) $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 4: Shuffling shares (Algorithm 4): IT analysis.

(e.g., because of sufficiently noisy leakages as we assume for now), she succeeds for a share with probability $\frac{1}{\eta}$, and so with probability $\left(\frac{1}{\eta}\right)^d$ for the d shares. As a result, the shuffling shares method can be interpreted as providing an increase of the noise on each share by a factor equal to the permutation size $\pi = \eta$. Masking then amplifies this emulated noise exponentially. The impact of this combination on the (worst-case) attack data complexity is therefore given by:

$$N^{\text{shares}} \geq \frac{cst}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})/\eta} = \frac{cst \cdot \eta^d}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})} \cdot \tag{16}$$

This equation is confirmed by the IT analysis of Algorithm 4 in Figure 4. Indeed for large noise, the $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ of the shuffled shares implementation is η^d times lower than the $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ of the masked only implementation. For example, for $\eta = 4$ and $d = 2$, this ratio is of $4^2 = 16$. For $\eta = 3$ and $d = 3$, this ratio is of $3^3 = 27$. Therefore, by using d permutations of size $\pi = \eta$, this solution provides an exponential amplification of the noise emulated thanks to shuffling. For completeness, we report similar results by assuming a leaking permutation in Appendix A, Figure 13. Slightly more noise is required to hide information on the permutation indexes, but we keep the same asymptotic improvement.

Shuffling everything. The two previous options shuffled either over the shares or over the tuples. A last solution is to shuffle jointly all the shares corresponding to all the pairs (i, s) by using a single permutation on $\pi = d \cdot \eta$ elements. This combination is illustrated in Figure 2c and formally defined in Algorithm 5.

In terms of side-channel security, recovering information about a secret element a_s without knowledge of the permutation (e.g., because of sufficiently noisy leakages) now requires that the adversary exploits the leakage of all the d shares

Algorithm 5 Masking and shuffling everything.

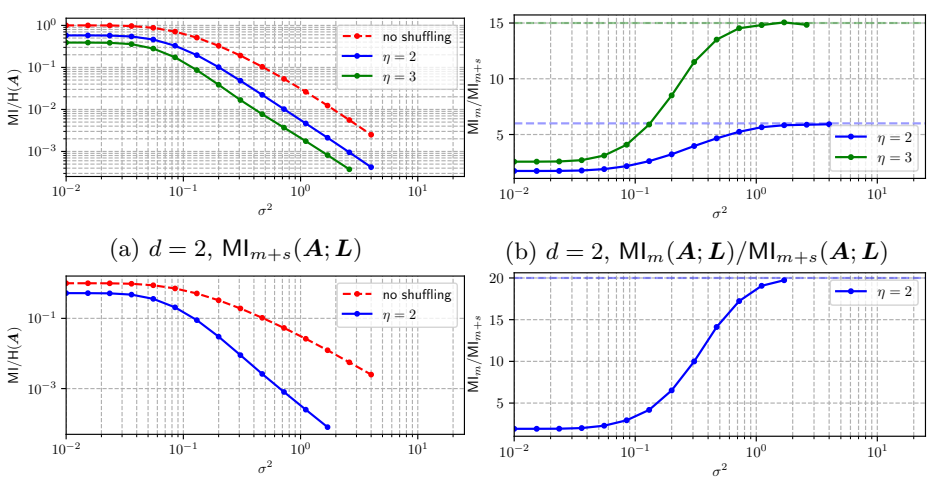
Input: $a_j = \sum_{i=0}^{d-1} a_j^i$ and $\mathsf{op}_{\mathsf{I}}(\cdot)$
Output: $b_j = \mathsf{op}_{\mathsf{I}}(a_j) = \sum_{i=0}^{d-1} b_j^i$.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Theta}$ ▷ Perm. generation of size $\pi = d \cdot \eta$
for c in $[0, \ldots, (d \cdot \eta) - 1]$ **do**
  $(i, j) \leftarrow (\theta_c \mod d, \theta_c // d)$
  $b_j^i = \mathsf{op}_{\mathsf{I}}(a_j^i)$

a_s^i. This implies to find the cycles c where the d pairs (i, s) with $0 \leq i < d$ are used. To do so, she chooses the first share out of the set of $d \cdot \eta$ shuffled values. Since d of these values correspond to a share a_s^i, she succeeds with probability $\frac{d}{d \cdot \eta}$. The second share can be guessed correctly with probability $\frac{d-1}{d \cdot \eta - 1}$ by excluding the first selected share. Eventually, the adversary can obtain information on a_s if she selects correctly all the d shares which happens with probability $\prod_{i=0}^{d-1} \frac{d-i}{d \cdot \eta - i} = \frac{1}{\binom{d \cdot \eta}{d}}$. Hence, the attack data complexity is growing as:

$$N^{\text{everything}} \geq \frac{cst \cdot \binom{d \cdot \eta}{d}}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})} \cdot \tag{17}$$

Once again, we confirm this equation with an IT analysis of a simulated implementation of Algorithm 5. For large noise, the ratio $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L}) / \mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ is equal to $\binom{2 \cdot 2}{2} = 6$ for $d = 2$ and $\eta = 2$, equal to $\binom{2 \cdot 3}{2} = 15$ for $d = 2$ and $\eta = 3$ and finally equal to $\binom{3 \cdot 2}{3} = 20$ for $d = 3$ and $\eta = 2$. This confirm the combinatorial security amplification of this combination of shuffling and masking.



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L}) / \mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (d) $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L}) / \mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 5: Shuffling everything (Algorithm 5): IT analysis.

Discussion. We conclude from the previous analyses and experiments that $N^{\text{tuples}} \leq N^{\text{shares}} \leq N^{\text{everything}}$. However, these successive improvements come at the cost of more or larger permutations. Hence, they raise the question of which option provides the best cost vs. security tradeoff, which we will discuss in section 5. For this purpose, a preliminary is to generalize our results from linear operations to non-linear ones, which we tackle in the next section. We note that Rivain et al. used the *shuffling everything* method for their linear layers (and derived a correlation-based bound for this purpose), but they only used the *shuffling tuples* one for the AES S-boxes [34]. To the best of our knowledge, the *shuffling shares* method is new. As will be shown next, it is the one leading to the best cost vs. security tradeoff. In particular, it allows avoiding the imbalance between the security levels of linear and non-linear operations, which is caused by the use of two types of shuffling, and forced Rivain et al. to artificially increase the permutation size of the non-linear layers by using dummy operations.

4.2 Non-linear operations

The second building block for d-probing secure circuits are non-linear operations. We will consider the standard ISW multiplication for this purpose.[10] In this case, all the pairs (i, j) have to be accessed in order to compute the cross products $a^i \otimes b^j$. We assume a setting where η independent ISW multiplications $c_s = a_s \otimes b_s$, with $0 \leq s < \eta$, have to be computed. In order to perform such operations, it implies that all the triplets (s, i, j) have to be accessed to compute all the $a_s^i \otimes b_s^j$ cross-products. Next, we list different ways to shuffle the computation of these triplets and discuss the different security levels they lead to. We note that such combinations of masking and shuffling for non-linear layers are more complex than for linear ones: there are more possibilities to be considered and their security is sometimes hard to assess. We therefore start by presenting the simplest solution of *shuffling tuples* used by Rivain et al. We then introduce a generic taxonomy of shuffling configurations which allows describing the design space of the masking + shuffling combinations, and we illustrate this taxonomy with the shuffling tuples approach. Finally, we prune this design space and focus on a number of solutions that can be viewed as the counterparts of the *shuffling shares* and *shuffling everything* approaches previously described for linear operations. We also argue why this pruning is practically relevant.

Shuffling tuples. This first method is similar to the *shuffling tuples* for linear layers. It is presented in Algorithm 6 where only the accesses to the tuples $\boldsymbol{a}_s$ and $\boldsymbol{b}_s$ are shuffled with a permutation of size $\pi = \eta$. Then the pairs (i, j) are accessed deterministically within these tuples.[11] Similarly to the shuffling tuples for linear layers, this allows an adversary to obtain information on the unshared secrets

[10] Our conclusions apply to other masking schemes based on a similar structure.

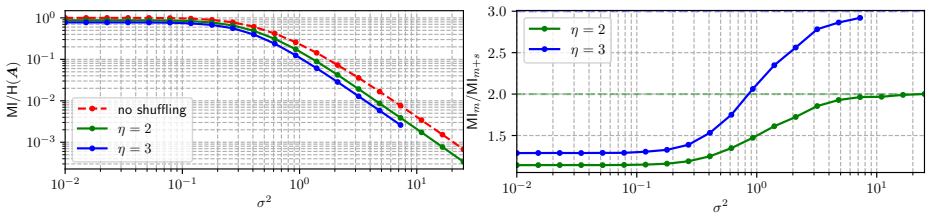
[11] The sequence of operations in Algorithm 6 are slightly different than in Algorithm 1. Namely both i and j range from 0 to $d-1$ and the first loop on $a^i \otimes b^i$ is omitted. The constraints on $r_s^{i,j}$ make these two algorithms equivalent.

Algorithm 6 Shuffling tuples ISW (configuration $(1^s, 0^i, 0^j)$).

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.
Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta-1\}, c_s = a_s \otimes b_s$.

```
θ^s ← Θ^s                                   ▷ permutation over η elements
for s in θ^s do
    for i in [0, ..., d − 1] do
        for j in [0, ..., d − 1] do
            c_s^i ← c_s^i ⊕ r_s^{i,j} ⊕ (a_s^i ⊗ b_s^j)
```

a_s, b_s and c_s with probability $\frac{1}{\eta}$. The resulting security guarantee is then linear with the size of the permutation similarly to Equation 15. This is confirmed by our IT analysis of Algorithm 6 reported in Figure 6. There, the ratio between the MI of a masked only ISW multiplication $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ and the one of shuffled and masked ISW multiplication $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ is always equal to η. This combination is a straightforward adaptation of the Rivain et al. implementation of masked and shuffled AES SubBytes to the case of ISW multiplications.



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 6: Shuffling ISW multiplication tuples (Algorithm 6): IT analysis.

Shuffling more. As in the previous subsection, the next step is to shuffle over more operations in order to reach a better security gain. For example, one natural goal would be to amplify the effect of shuffling with masking so that a gain factor of η^d can be maintained for full implementations. For this purpose, we first describe all the possible shuffling and masking combinations which handle the indexes s, i and j independently, with the next shuffling configurations.

Shuffling + masking configurations. We describe all the possible combinations of masking and shuffling with Algorithm 7 along with a configuration of the form $(x^\alpha, x^\beta, x^\gamma)$. The algorithm is composed of three nested loops where each loop is responsible for shuffling (or not) one of the indexes s, i or j. In the configuration, the superscripts correspond to the index manipulated by the loop and the position in the triplet corresponds to the order of the loops. For example, the first superscript designates the outermost loop and the third one the innermost loop.

Additionally, the x value is a bit set to 1 if the loop is shuffled and 0 otherwise. We note that swapping the indexes i and j leads to the same gadget since the multiplication is commutative. As a result, the configuration $(0^s, 0^i, 0^j)$ denotes an unshuffled implementation, and the previous *shuffling tuples* solution given in Algorithm 6 corresponds to the configuration $(1^s, 0^i, 0^j)$, where only the outer loop on s is shuffled. The Greek letters in Algorithm 7, namely $\alpha \in A, \beta \in B$ and $\gamma \in \Gamma$, are uniquely set to s, i or j. The corresponding capital letter is the set of values that the index can take. For example, if $\alpha = s$ then $A = [0, \ldots, \eta - 1]$ and if $\alpha = i$ or $\alpha = j$ then $A = [0, \ldots, d - 1]$. Hence, $\boldsymbol{\theta}^\alpha \xleftarrow{x^\alpha} \boldsymbol{\Theta}^\alpha$ is a fresh permutation of the elements in A if x^α is set to one and is A otherwise.

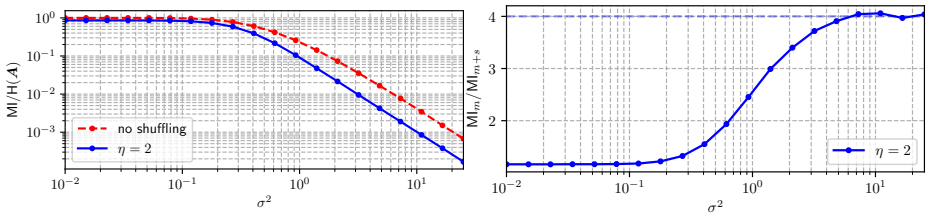
Algorithm 7 Generic shuffled & masked ISW multiplications.

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$, shuffling configuration $(x^\alpha, x^\beta, x^\gamma)$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.
Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta - 1\}, c_s = a_s \otimes b_s$.

```
for α in θ^α ←(x^α)— Θ^α do                    ▷ one permutation over A
    for β in θ^β ←(x^β)— Θ^β do                 ▷ |A| permutations over B
        for γ in θ^γ ←(x^γ)— Θ^γ do             ▷ |A| · |B| permutations over Γ
            c_s^i ← c_s^i ⊕ r_s^{i,j} ⊕ (a_s^i ⊗ b_s^j)
```

Pruning the configuration space. Based on the previous configurations, there are 6 possible ways to order the loops and, for each of these orderings, there are 2^3 possibilities of shuffling. This leads to a total of 48 cases. In order to reduce the number of configurations to investigate in detail, we first notice that in Algorithm 7, shuffling the indexes i or j never leads to a security improvement. Indeed, shuffling on i (resp., j) only means shuffling the accesses to the shares a_s^i (resp., b_s^j). Because in an encoding $\boldsymbol{a}_s$ of a_s, the position of the shares has no effect on security, an adversary can obtain information on a_s by observing leakages on all the shuffled shares a_s^i without being impacted by the shuffling of i (resp., j). We further note that in the high-noise regime (that we assume for now), this observation also holds when we shuffle i and j. In this case, the cross-products are shuffled and information on each of the a_s^i involved in these cross-products is obtained from the η observations of the shuffled $a_{\boldsymbol{\theta}^s}^i$. But information on a_s can still be recovered from the input/output tuples of the multiplication, without being impacted by the shuffling of i. So this configuration makes the information of the cross-products harder to exploit, while it is known that the information provided by these cross-products is dominated by the information of the multiplications' input/output tuples when the noise is large [13]. Therefore, we next limit our investigations to configurations with 0^i and 0^j.

This reduces the set of combinations to 3 possibilities. We next list them and discuss their security. The first one is $(1^s, 0^i, 0^j)$ and corresponds to the aforementioned *shuffling tuples* (Algorithm 6) for which the security impact is only linear in the size of the permutation. The second configuration is $(0^i, 1^s, 0^j)$ (resp., $(0^j, 1^s, 0^i)$). For this configuration the security of the inner loop variable b^j (resp., a^i) differs from the one on the outer loop variable a^i (resp., b^j) and the security of the inner loop variable is similar to the *shuffling tuples* option, which is not desirable. Finally, the third configuration is $(0^i, 0^j, 1^s)$. It is similar to the *shuffling shares* of linear layers, where the permutation is applied to $\pi = \eta$ independent elements. In this case, every loading of the input shares a_s^i and b_s^j as well as the updates of c_s^i are shuffled among the η independent ISW multiplications. As a result, the information on each of these operations is reduced by a factor η that is later amplified by masking. Therefore, it provides an exponential gain factor η^d as in Equation 16. This gain is confirmed by the IT analysis depicted in Figure 7 for $d = 2$ and $\eta = 2$, where the ratio between $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ and $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ for high enough noise is of $2^2 = 4$.



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ (b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 7: Shuffling ISW shares (Algorithm 7, config. $(0^i, 0^j, 1^s)$): IT analysis.

Shuffling everything-like. A last natural question is to know whether it is possible to obtain a better security, similar to the one obtained for the *shuffling everything* of linear layers. For these operations, the improvement was obtained by shuffling jointly all the pairs (i, j) instead of independently. For the ISW multiplications, this can be done by merging some of (or all) the loops in Algorithm 7. To list the possible combinations when two loops are merged, we use Algorithm 8 along with notations of the form $(x^{\alpha,\beta}, x^{\gamma})$. For example, $(x^{i,j}, x^s)$ means that the outer loop is a merge of the loops on i and j and hence over d^2 elements. More precisely, in Algorithm 7 the operator $\times$ denotes the Cartesian product and $\boldsymbol{\theta}^{\alpha,\beta}$ is a permutation over the set $A \times B$. Alternatively, all loops can be merged together into a single one operating on $d^2 \cdot \eta$ elements. We denote this configuration as $(x^{s,i,j})$ and detail it in Appendix B, Algorithm 9.

Based on these configurations, a first observation is that loops on i and j must be merged together in order to avoid the same asymmetry issues as in the previous *shuffling shares* approach. This reduces the possibilities of merging

Algorithm 8 Generic shuffled masked ISW multiplications with 2 loops merged.

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$, shuffling configuration $(x^{\alpha,\beta}, x^{\gamma})$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.
Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta-1\}, c_s = a_s \otimes b_s$.

for α, β in $\theta^{\alpha,\beta} \xleftarrow{x^{\alpha,\beta}} \boldsymbol{\Theta}^{\alpha,\beta}$ **do**
  for γ in $\theta^{\gamma} \xleftarrow{x^{\gamma}} \boldsymbol{\Theta}^{\gamma}$ **do** ▷ $|A|$ permutations over B
    $c_s^i \leftarrow c_s^i \oplus r_s^{i,j} \oplus (a_s^i \otimes b_s^j)$

loops to three combinations that are $(1^s, 1^{i,j})$, $(1^{i,j}, 1^s)$ and $(1^{s,i,j})$. However, all these options induce a non-uniform permutation of the outputs shares, which due to the fact that the output share c_0^0 is valid once it has been updated d times in Algorithm 8. Therefore, during the first few iterations it is unlikely that the current operation is the d-th update of c_0^0 as it becomes for the last few iterations. This effect is illustrated in the plots of Figure 14 given in Appendix C, where the probability to generate c_0^0 globally increases with t, with some differences depending on the shuffling configurations. Such non-uniform permutations of the output shares may offer some (even good) level of security. Yet, they suffer from the drawback that their analysis requires analyzing complex permutation biases that are specific to the configurations and the parameters d and η. We therefore rule out these options in our following investigations for usability reasons.

4.3 Summary of our investigations

Table 1 contains a summary of the different combinations of masking and shuffling we considered, when the secret vectors are of size η. The table reports the security gain factor compared to a masked-only implementation (i.e., $\mathsf{MI}_m/\mathsf{MI}_{m+s}$), the size of the permutation(s) used (i.e., π) and the number of fresh permutation(s) needed (i.e., # perm.). As mentioned above, the *shuffling everything* approach cannot be straightforwardly applied to non-linear layers. Hence, we next focus on the *shuffling shares* solution which allows the same exponential security gain for both linear and non-linear layers, enabling balanced designs (and evaluate the *shuffling tuples* option for comparison purposes).

	Linear layer			Non-linear layer		
	$\mathsf{MI}_m/\mathsf{MI}_{m+s}$	π	# perm.	$\mathsf{MI}_m/\mathsf{MI}_{m+s}$	π	# perm.
shuffling tuples	η	η	1	η	η	1
shuffling shares	η^d	η	d	η^d	η	d^2
shuffling everything	$\binom{d \cdot \eta}{d}$	$d \cdot \eta$	1	?	?	?

Table 1: Summary of the shuffling + masking combinations.

5 Time versus security for shuffled ISW

In the previous section, we have discussed how to amplify the impact of shuffling by combining it with masking, for both linear and non-linear operations. In this section, we focus on the practical instantiation of the *shuffling tuples* and *shuffling shares* methods, with a focus on 32-bit software platforms, and we compare them to a masked-only implementation. That is, we study Algorithm 7 with configurations $(1^s, 0^i, 0^j)$, $(0^i, 0^j, 1^s)$ and $(0^s, 0^i, 0^j)$ in the context of bitslice software implementations. We first detail the randomness that is required by such implementations. Then, we describe the parameters that influence their execution time and their security level. Finally, we propose concrete performance evaluations and leverage them in order to discuss general guidelines for combinations of masking and shuffling that best trade performance and security.

As a preliminary remark, we mention that when protecting a cryptographic implementation with suffling, a preliminary (cipher-specific) challenge is to find independent operations that can be executed in parallel. In order to keep the following discussions independent of the primitive to protect, we therefore consider a general use case where we aim to implement #AND ISW multiplications in parallel. As will be clear next, it allows us to draw general conclusions about how masking and shuffling should be combined (i.e., which of the countermeasures should use the available parallelism in priority), which can then be directly translated into concrete guidelines for implementing actual ciphers (or possibly modes of operations, in case they offer additional levels of parallelism).

5.1 Randomness requirements

The randomness required to combine masking and shuffling is composed of two terms. The first one is due to masking and remains independent of the shuffling parameter η. For the masked (only) ISW multiplication from Algorithm 1, $d \cdot (d-1)/2$ random bits per bitwise multiplication are needed. This means that #AND times more random bits are needed in our context. The second term is due to shuffling and corresponds to the randomness needed to generate the permutation(s) $\boldsymbol{\theta}$. Precisely, a permutation on η elements can be generated with $\eta \cdot \log_2 \eta$ random bits [39]. When combined with masking by *shuffling tuples*, a single of these permutations must be generated, as reported in Table 1. When combined with masking by *shuffling shares*, a total of d^2 of these permutations must be generated. As a result, the *shuffling shares* strategy requires a total amount of random bits given by:

$$\underbrace{\left(\#\text{AND} \cdot \frac{d \cdot (d-1)}{2}\right)}_{\text{masking}} + \underbrace{\left(d^2 \cdot \eta \cdot \log_2 \eta\right)}_{\text{shuffling}} . \tag{18}$$

5.2 The bitslice masking + shuffling design space

We now introduce the different parameters that influence the execution time as well as the security level of masked and shuffled bitslice implementations. These

parameters are summarized in Table 2, where the first block contains parameters that are under control of the implementers and the second one contains parameters that depend (partially) on the software platform used.

Parameter	Description
#AND	Number of masked bitwise AND to operate
bs	Effective size of the registers
$\pi = \eta$	Size of the permutations such that $\eta = \#\text{AND}/bs$
d	The number of shares
r	Number of cycles to generate 32 random bits
$\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$	Mutual information on a single unshuffled shared bit

Table 2: Summary of parameters for bitslice masking and shuffling.

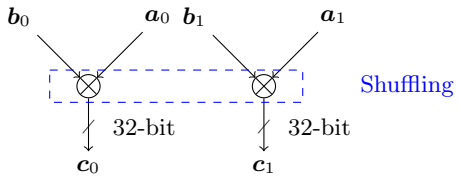
We next detail these parameters and their interactions.

First, we recall that bitslicing is a software programming technique that represents an algorithm (e.g., a block cipher) with Boolean operations [8]. It takes advantage of the parallelism enabled by bitwise instructions which are available in most (all) the modern micro-controllers (MCU). For example, it is possible to place 32 bits in a register $\boldsymbol{a}$, 32 bits in a register $\boldsymbol{b}$, and to obtain the 32 bits corresponding to $\boldsymbol{a} \oplus \boldsymbol{b}$ in a single instruction. This strategy is particularly appealing in the context of masking where multiple bitwise ISW multiplications can be applied in parallel. Several works have shown how it can be efficiently used to protect block cipher implementations at arbitrary security orders [20,6,5].

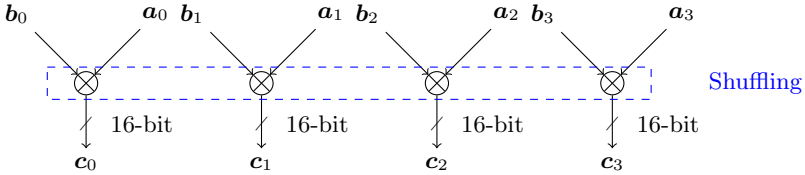
Next, we observe that in contrast with bitslice implementations that favor parallelism, shuffling rather applies to serialized independent operations: the more such operations, the larger the size of the permutation and therefore the impact of shuffling. As a result, the main question when combining bitslice masking and shuffling is whether one should favor serialization or parallelism.

In order to answer this question, we will analyze the impact of the "effective size" of the register, denoted as bs, which is the parameter reflecting the tradeoff between serialization and parallelism. It is defined as the number of useful bits in a single register, and therefore corresponds to the number of Boolean operations that are parallelized. The maximum value of bs is given by the physical register size. Therefore, on a 32-bit software platform, we have $bs \leq 32$. Yet, the designer could also reduce bs to increase the permutation size η that is equal to $\#\text{AND}/bs$. For example, Figure 8 illustrates two options in the masking + shuffling design space for #AND=64. The first option (Figure 8a) is to maximize the parallelism with $bs = 32$ and so $\eta = 2$. A second option (Figure 8b) is to decrease the parallelism with $bs = 16$ and to increase the permutation size up to $\eta = 4$.

The other parameters that we need to evaluate the performance vs. security tradeoff of masked and shuffled implementations are the randomness cost and the noise level. Precisely, and as described in subsection 5.1, both masking and shuffling require randomness. Hence, the throughput at which random bits are



(a) $bs = 32$, $\eta = 2$



(b) $bs = 16$, $\eta = 4$

Fig. 8: Different options in the design space for #AND=64.

generated has an impact on the cycle count. We introduce the parameter r that is the latency in cycles needed to generate 32 random bits once requested. As for the noise level, we quantify it with the amount of information that is available on a single share of an unprotected implementation $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$.

5.3 Performance evaluation and discussion

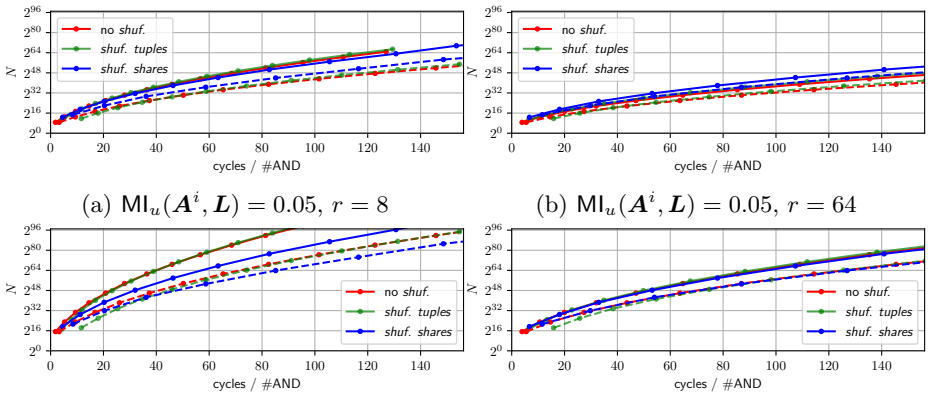
In order to evaluate the security vs. performance tradeoff of the *shuffling tuples* and *shuffling shares* options, we complement the previous security evaluations by measuring the total cycle count of protected implementations running on a 32-bit ARM Cortex-M4 with the design parameters from Table 2.[12] Precisely, when only masking we use a bitslice instantiation of Algorithm 1 that is repeated η times to operate #AND bitwise secure multiplications. The security level is then given by Equation 7. For the *shuffling tuples* strategy, a similar approach is used but the η ISW multiplications are performed out of order thanks to the generation of a single permutation. The security level is then given by Equation 15. For the *shuffling shares* strategy, we use an efficient implementation (Appendix B, Algorithm 10) that does not require to pre-compute randomness but rather generates it on-the-fly. Its security level is given by Equation 16.[13]

The resulting execution time versus security curves are reported in Figure 9 for #AND=128 and in Figure 10 for #AND=512. The x-axis is the number of

[12] Implementations are written in C and compiled with the -O3 optimization flag.

[13] We stress that the comparative value of our information theoretic analyses is due to the fact that the constants they imply are independent of the masking and shuffling parameters (i.e., they only depend on the size of the target intermediate variable and success rate). We also recall that they are only valid under a sufficient noise. The concrete investigation of a low-noise case study is given in the next section.

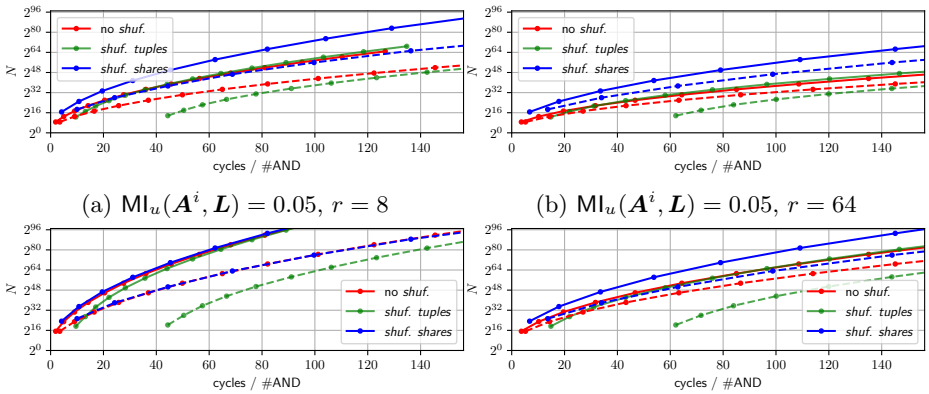
cycles used to perform the secure multiplications normalized by #AND. The y-axis reports the data complexity of the worst-case attack. Each data point is for a different masking order d, with the leftmost being $d = 2$ and increasing with steps of one when moving to the right. The red curves are for masked-only implementations, green ones for *shuffling tuples* and blue ones for *shuffling shares*. Continuous lines are for $bs = 32$, hence using the full physical register. The dashed curves are for $bs = 16$, hence doubling the serialization.



(a) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 8$ (b) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 64$

(c) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 8$ (d) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 64$

Fig. 9: Cycles vs. security for 128 bitwise ANDs for various noise levels and randomness cost. Continuous are for $bs = 32$ and dashed ones for $bs = 16$.



(a) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 8$ (b) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 64$

(c) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 8$ (d) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 64$

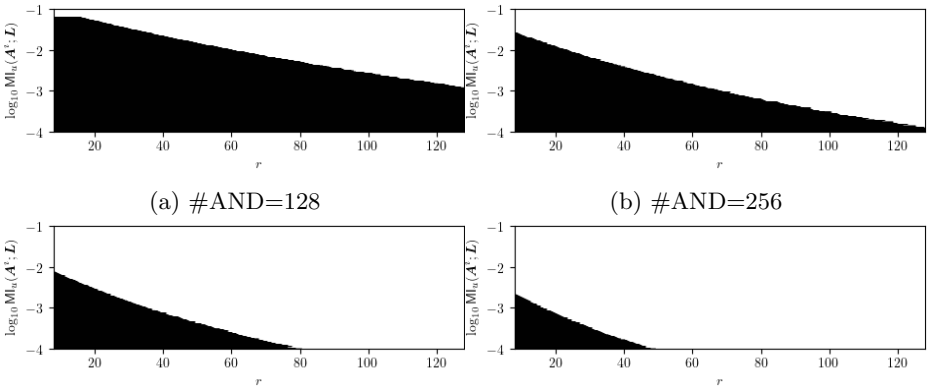
Fig. 10: Cycles vs. security for 512 bitwise ANDs for various noise levels and randomness cost. Continuous are for $bs = 32$ and dashed ones for $bs = 16$.

We draw two general conclusions regarding the combination of masking and shuffling from these plots. First, the choice of taking $bs = 32$ is always better than taking $bs = 16$. That is for a fixed execution time (x-axis), the resulting attack data complexity is always larger for $bs = 32$ than for $bs = 16$, and this trend holds for lower bs values. It implies that a designer should first favor the parallelization of the masked computations he has to perform, and shuffle what is left to serialize. It also confirms the relevance of bitslicing when combining masking and shuffling.[14] Second, the combination of masking and shuffling benefits from a larger #AND. For example, if 80 cycles are spend per bitwise multiplication, the resulting security is about 2^{40} for #AND=128 for *shuffling shares* according to Figure 9b. For the same number of clock cycles per bitwise multiplication, having #AND=512 provides a security around 2^{50} according to Figure 10b.

Masking only vs. shuffling tuples. By comparing the masked-only implementation with the *shuffling tuples* strategy, we observe that *shuffling tuples* does not bring a significant gain. That is, for a fixed performance level (i.e., value of the x-axis), *shuffling tuples* at best brings a marginal improvement. This can be explained by the fact that *shuffling tuples* is always slower than masking only due to the permutation generation, while only bringing a gain of η in attack complexity. Increasing this security gain would require to reduce bs, and so to favor serialization. But as shown above, this degrades the time vs. security tradeoff which rather pushes for masking with maximum parallelism.

Masking only vs. shuffling shares. Interestingly, the conclusion is more shaded when considering the *shuffling shares* approach. In this case, the interest of combining countermeasures essentially depends on the randomness cost r, noise level $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$ and amount of independent operations available #AND. That is, for expensive randomness, relatively low noise (still sufficient for the countermeasures to be effective) and high #AND, *shuffling shares* is the best solution. Whenever decreasing #AND or the randomness cost, or increasing the noise level, this advantage vanishes and masking-only can become the best option. (e.g., in Figure 9c). We summarize this design space with Figure 11, where the x-axis is the randomness cost and the y-axis is the noise level $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$. Black areas represent contexts where masking alone is more efficient than *shuffling shares* in order to reach a given security target (here 64-bit). White areas represent contexts where *shuffling shares* is more efficient. Black areas are for cheap randomness and high noise levels (and lie in the bottom left). By increasing the number of independent multiplications #AND (and accordingly η), this area is reduced and the masking + shuffling approach becomes beneficial.

[14] One corollary of these observations is that (for high enough noise levels as we consider), the use of dummy operations in order to increase the amount of operations to shuffle does not pay off: increasing the number of shares is a better strategy.



(a) #AND=128 (b) #AND=256

(c) #AND=512 (d) #AND=1024

Fig. 11: Masking + *shuffling shares* design space for a 64-bit security target and $bs = 32$. Masking only is the most efficient solution in black areas.

6 Real-world (low-noise) case study

Based on the previous evaluations, we can conclude that the new (*shuffling shares*) combination of masking and shuffling that we propose in this paper can indeed be an interesting asset for the design of side-channel secure implementations. It could in particular be quite effective to protect bitslice designs based on large permutations (e.g., Keccak [7]) where a lot of parallel operations can be identified. Yet, these evaluations have been performed under the assumption of a sufficient noise level which may not be observed in practice. In this section, we therefore complement these analyses with a real-world evaluation of the masking + shuffling countermeasure implemented in a commercial MCU.

The motivation for this investigation comes from the intuition that shuffling could be an option to protect devices where the physical noise is not sufficient for masking to be directly effective. In order to clarify whether this expectation is founded, we propose a worst-case evaluation where the adversary has full knowledge and control of her target implementation during a profiling phase. In particular, she has knowledge of the randomness used to generate the permutations enabling profiled attacks against the shuffling as in section 3. We conclude this section by briefly discussing how our observations evolve when relaxing the adversarial capabilities, in the spirit of a backwards evaluation [2].

6.1 Target and measurement setup

We investigate the implementation of ISW multiplications enhanced with the *shuffling shares* approach, executed on the 32-bit ARM Cortex-M3 of the STM32 VLDISCOVERY commercial board. In order to obtain clean measurements, we modified the board and removed all the decoupling inductances and capacitors.

We used the available slot to add an 8[MHz] external crystal and derived the maximum 24[MHz] internal clock from it. In order to measure the side-channel leakage $\boldsymbol{L}$, we placed a current probe (i.e., the CT1 from Tektronix 1[GHz]) on the dedicated jumper between the on-board power regulator and the MCU power pins. Eventually, we sampled this signal with a 12-bit resolution at a sampling rate of 500[MSamples/s] thanks to a PicoScope 5244D. As shown next, the resulting setup allows collecting measurements with low noise.

6.2 Worst-case adversary

We evaluated this implementation with the adversary of section 3 using the model $\tilde{\mathsf{m}}_{\mathsf{New}}(\boldsymbol{y}|\boldsymbol{l})$ from Equation 13. We extract information on the secret variables using a template attack in a principal subspace according to the methodology of [10,11]. Precisely, we estimate the leakage PDFs with Gaussian templates after dimensionality reduction using linear discriminant analysis [1].

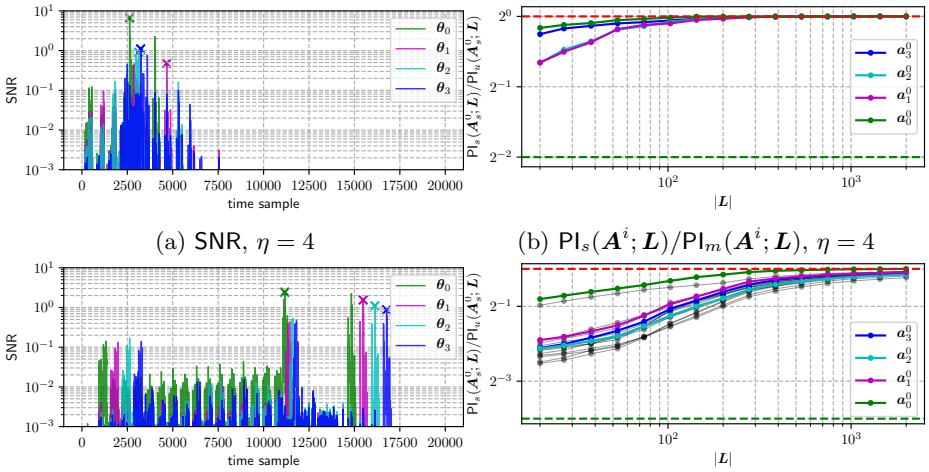
The resulting leakage analysis is reported in Figure 12 for implementations with $\eta = 4$ and $\eta = 16$. On the left figures, the x-axis is the time and the y-axis is the SNR (in log-scale) of the indexes of the permutation $\boldsymbol{\theta}_c$. We observe that the maximum value for each of them is around 1 and many other dimensions lead to an SNR two orders of magnitude lower (especially for $\eta = 16$). On the right, we report the number of dimensions of the leakage vector exploited by the adversary $|\boldsymbol{L}|$ on the x-axis and the y-axis reports (in log-scale) the information reduction per share. Namely it is the ratio between the PI of elements in the first share vector $\boldsymbol{a}^0$ for a shuffled implementation (i.e., $\hat{\mathsf{PI}}_s(\boldsymbol{A}_c^0; \boldsymbol{L})$) and the one of an unshuffled implementation (i.e., $\hat{\mathsf{PI}}_u(\boldsymbol{A}_c^0; \boldsymbol{L})$). The green dashed curves represents the expected impact of the shuffling in case permutation indexes do not leak (as assumed in section 4), namely $1/\eta$. The red dashed curve equals 1 and corresponds to a level of leakage such that shuffling is ineffective.[15]

As the number of dimensions exploited by the adversary increases, this ratio gets closer to one. For $\eta = 4$, it is stuck at 1, meaning that the adversary directly gains full knowledge of the permutation and is therefore not impacted by the shuffling. For $\eta = 16$, a similar behavior is observed but a larger number of dimensions must be exploited. For $|\boldsymbol{L}| = 2,000$, the average ratio is equal to 0.92, meaning that the information per share is only reduced by 8%. For example, it implies that the gain factor of this combination of countermeasures with 8 shares is around 2 (it would be 16^8 with a sufficient noise).[16] We conclude that this implementation lies in the low noise region of Figure 13 in Appendix A, where a limited gain is obtained by combining masking & shuffling.

We note that these conclusions are based on a worst-case attack strategy. Non-profiled attacks unable to characterize (and therefore exploit) the permutation leakages may lead to a more positive conclusion (i.e., force the adversary

[15] We report also the PI on the permutation indexes in Appendix D, Figure 15.

[16] We refer to [11], Figure 16, for details about the impact of such an information reduction factor on the worst-case security of bitslice masked implementations.



(a) SNR, $\eta = 4$ (b) $\mathsf{PI}_s(\boldsymbol{A}^i;\boldsymbol{L})/\mathsf{PI}_m(\boldsymbol{A}^i;\boldsymbol{L})$, $\eta = 4$

(c) SNR, $\eta = 16$ (d) $\mathsf{PI}_s(\boldsymbol{A}^0_s;\boldsymbol{L})/\mathsf{PI}_m(\boldsymbol{A}^0_s;\boldsymbol{L})$, $\eta = 16$

Fig. 12: IT analysis of real permutation leakages. Left: SNR in function of the time. Right: information loss in function of the # of dimensions exploited.

to sum the noise variances of her shuffled operations). Yet, assuming that only such weaker attacks are possible appears as a risky strategy in view of recent progresses (e.g., using deep learning) where a worst-case information extraction is approached with limited knowledge of the target implementation [28]. These experiments also confirm the interest of applying shuffling to large designs (e.g., permutation-based), since large η values make the attack more difficult.

Eventually, we insist that these experiments do not show that masking and shuffling cannot be implemented securely (the previous sections showed the opposite). What they show is that when the noise level provided by a leaking device is too low for masking to be effective, it is in general too low for shuffling to be effective as well. Indeed, bitslice masking on similar MCUs generally requires a large number of shares [11]. It may even be more challenging to implement shuffling securely on low-cost embedded devices since the memory access it relies on can be more leaky than (for example) bitslice computations.

So overall, it remains an important challenge to ensure a sufficient level of noise on embedded MCUs, so that masking, shuffling and their combination becomes effective. Reaching this goal with existing low-cost devices (similar to the ARM Cortex-M3 we analyzed) is an interesting research direction. Yet, the recurrent difficulties caused by low physical noise for the implementation of side-channel countermeasures also suggest that solving the issue at the technological level by guaranteeing a minimum level of intrinsic noise on security MCUs could be highly beneficial in terms of security vs. performance tradeoff.

Acknowledgments. François-Xavier Standaert is a senior associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the ERC project 724725 (acronym SWORD).

References

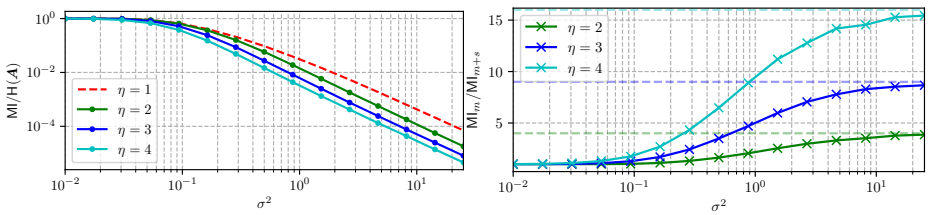
1. C. Archambeau, E. Peeters, F. Standaert, and J. Quisquater. Template attacks in principal subspaces. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.
2. M. Azouaoui, D. Bellizia, I. Buhan, N. Debande, S. Duval, C. Giraud, É. Jaulmes, F. Koeune, E. Oswald, F. Standaert, and C. Whitnall. A systematic appraisal of side channel evaluation strategies. In *SSR*, volume 12529 of *Lecture Notes in Computer Science*, pages 46–66. Springer, 2020.
3. M. Azouaoui, F. Durvaux, R. Poussier, F. Standaert, K. Papagiannopoulos, and V. Verneuil. On the worst-case side-channel security of ECC point randomization in embedded devices. In K. Bhargavan, E. Oswald, and M. Prabhakaran, editors, *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, volume 12578 of *Lecture Notes in Computer Science*, pages 205–227. Springer, 2020.
4. J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F. Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
5. S. Belaïd, P. Dagand, D. Mercadier, M. Rivain, and R. Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In *EUROCRYPT (3)*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.
6. S. Belaïd, D. Goudarzi, and M. Rivain. Tight private circuits: Achieving probing security with the least refreshing. In *ASIACRYPT (2)*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
7. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.
8. E. Biham. A fast new DES implementation in software. In E. Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.
9. O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky, and F. Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.
10. O. Bronchain and F. Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.
11. O. Bronchain and F.-X. Standaert. Breaking masked implementations with many shares on 32-bit software platforms or when the security order does not matter. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):202–234, 2021.
12. N. Bruneau, S. Guilley, Z. Najm, and Y. Teglia. Multivariate high-order attacks of shuffled tables recomputation. *J. Cryptol.*, 31(2):351–393, 2018.

13. G. Cassiers and F. Standaert. Towards globally optimized masking: From low randomness to low noise rate or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):162–198, 2019.
14. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
15. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
16. J. Coron and L. Spignoli. Secure shuffling in the probing model. *IACR Cryptol. ePrint Arch.*, 2021:258, 2021.
17. E. de Chérisey, S. Guilley, O. Rioul, and P. Piantanida. Best information is most successful mutual information and success rate in side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):49–79, 2019.
18. A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.
19. A. Duc, S. Faust, and F. Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.
20. D. Goudarzi and M. Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.
21. V. Grosso and F. Standaert. Masking proofs are tight and how to exploit it in security evaluations. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 385–412. Springer, 2018.
22. T. Güneysu and A. Moradi. Generic side-channel countermeasures for reconfigurable devices. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2011.
23. C. Herbst, E. Oswald, and S. Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.
24. Y. Ishai, A. Sahai, and D. A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
25. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
26. S. Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.
27. S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
28. L. Masure, C. Dumas, and E. Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.
29. S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.*, 24(2):292–321, 2011.
30. S. Patranabis, D. B. Roy, A. Chakraborty, N. Nagar, A. Singh, D. Mukhopadhyay, and S. Ghosh. Lightweight design-for-security strategies for combined countermeasures against side channel and fault analysis in iot applications. *J. Hardw. Syst. Secur.*, 3(2):103–131, 2019.

31. S. Patranabis, D. B. Roy, P. K. Vadnala, D. Mukhopadhyay, and S. Ghosh. Shuffling across rounds: A lightweight strategy to counter side-channel attacks. In *ICCD*, pages 440–443. IEEE Computer Society, 2016.
32. E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
33. E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
34. M. Rivain, E. Prouff, and J. Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.
35. F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
36. F. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The world is not enough: Another look on second-order DPA. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.
37. M. Tunstall, C. Whitnall, and E. Oswald. Masking tables - an underestimated security risk. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*, pages 425–444. Springer, 2013.
38. N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Soft analytical side-channel attacks. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.
39. N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F. Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.

Supplementary Material

A Additional simulations with permutation leakage



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Fig. 13: Shuffling shares (Algorithm 4) with permutation leakage IT analysis.

B Additional algorithms

Algorithm 9 Generic shuffled masked ISW multiplications with all loops merged.

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\pi-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\pi-1}\}$, shuffling configuration $(x^{\alpha,\beta,\gamma})$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.
Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\pi-1}\}$ such that $\forall s \in \{0, 1, \ldots, \pi-1\}, c_s = a_s \otimes b_s$.

for α, β, γ in $\theta^{\alpha,\beta,\gamma} \xleftarrow{x^{\alpha,\beta,\gamma}} \boldsymbol{\Theta}^{\alpha,\beta,\gamma}$ **do**
  $c_s^i \leftarrow c_s^i \oplus r_s^{i,j} \oplus (a_s^i \otimes b_s^j)$

Algorithm 10 Efficient Masked and Shuffled ISW.

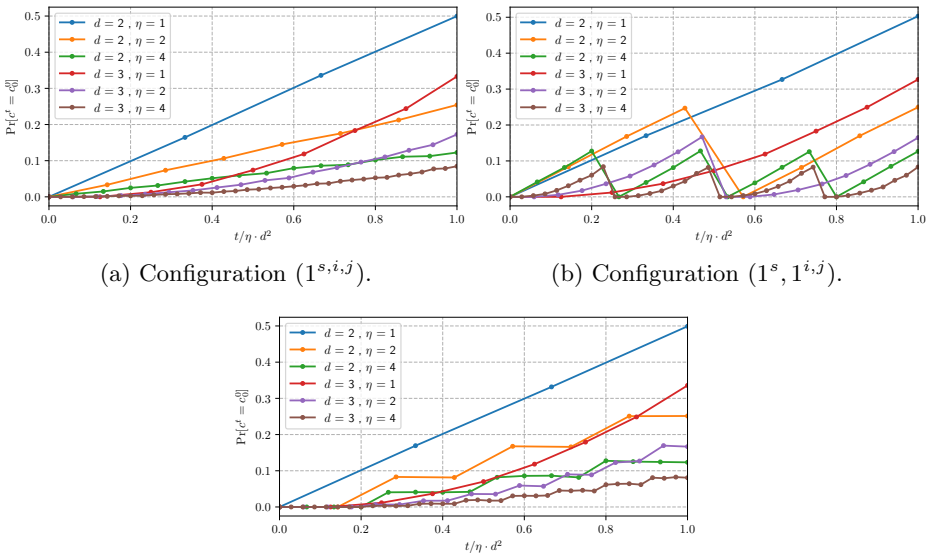
Input: $\boldsymbol{a}$ with $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\boldsymbol{b}$ with $b_s = \sum_{i=0}^{d-1} b_s^i$ and $\pi = |\boldsymbol{a}| = |\boldsymbol{b}|$.
Output: $\boldsymbol{c}$ with $c_s = \sum_{i=0}^{d-1} c_s^i$ and $c_s = a_s \otimes b_s$.

for i in $[0, \ldots, d-1]$ **do**
  for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
    $c_s^i = a_s^i \otimes b_s^i$
for i in $[0, \ldots, d-1]$ **do**
  for j in $[i+1, \ldots, d-1]$ **do**
    for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
      $r_s \leftarrow \{0,1\}$
      $c_s^i \leftarrow c_s^i \oplus (r_s \oplus (a_s^i \otimes b_s^j))$
    for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
      $c_s^j \leftarrow c_s^j \oplus (r_s \oplus (a_s^j \otimes b_s^i))$

C Shuffling everything-like non uniformities

Figure 14 illustrates the non-uniformity of the permutations in the shuffling everything-like context of section 4.2, for various parameters d and η.

The y-axis is the probability that a given valid output share (in this case c_0^0) is generated during the t-th iteration of the algorithm. The x-axis is the iteration t that is normalized by the total number of iterations $\eta \cdot d^2$.



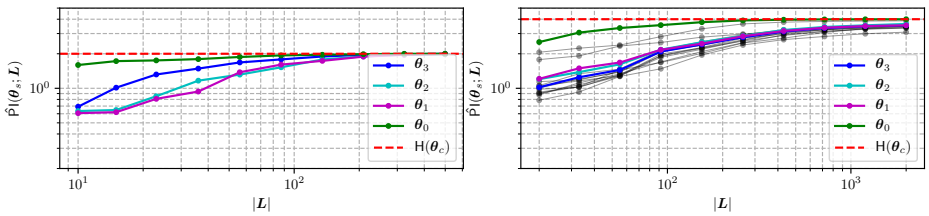
(a) Configuration $(1^{s,i,j})$.

(b) Configuration $(1^s, 1^{i,j})$.

(c) Configuration $(1^{i,j}, 1^s)$.

Fig. 14: Proba. to generate output share c_0^0 at cycle t when merging ISW loops.

D Perceived information on permutation indexes



(a) $\mathsf{PI}(\boldsymbol{\theta}_c; \boldsymbol{L})$, $\eta = 4$ (b) $\mathsf{PI}(\boldsymbol{\theta}_c; \boldsymbol{L})$, $\eta = 16$

Fig. 15: Information on $\boldsymbol{\theta}_i$ in function of the # of dimensions exploited.