Noname manuscript No.
(will be inserted by the editor)

Reducing Risks Through Simplicity

(High Side-channel Security for Lazy Engineers)

Olivier Bronchain[1] ✉ · Tobias Schneider[2] · François-Xavier Standaert[1]

the date of receipt and acceptance should be inserted later

Abstract Countermeasures against side-channel attacks are in general expensive, and a lot of research has been devoted to the optimization of their security vs. performance trade-off. Besides, a wide literature has also shown that implementing such countermeasures is an error-prone task and requires to deal with various engineering challenges (e.g., physical defaults, compositional errors, ...). This work aims to contribute on this second item, by evaluating the extent to which (almost) key-homomorphic primitives, and in particular a recent PRF instance based on the Learning With Rounding (LWR) problem, can lead to easy-to-implement and easier-to-evaluate side-channel secure designs. We confirm these properties by describing an FPGA implementation that does not require complex (compositional) reasoning in its analysis, can be masked securely under simple design conditions, and for which the evaluation directly scales to arbitrary number of shares. We provide a comprehensive performance and (worst-case) security analysis of our design, and compare the obtained results with those of an AES implementation protected with the Domain Oriented Masking (DOM) scheme. Results show that simplicity has a cost, which becomes less prohibitive as security requirements increase

Keywords Side-Channel Analysis · Masking · Worst-Case Evaluations · Key-Homomorphic PRFs · Learning With Rounding · FPGA implementations

[1] ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium ·
[2] NXP Semiconductors Austria, Gratkorn, Austria (The majority of the author's contribution was performed while he was with [1]) ·
✉ olivier.bronchain@uclouvain.be

1 Introduction

Side-Channel Attacks are important threats to the security of embedded systems. They exploit physical information leaked from implementations through, for example, their power consumption [41] or electromagnetic radiations [31] in order to recover secret information such as encryption keys. A standard approach to circumvent this threat is the masking countermeasure [17]. Its underlying principle is to split the sensitive data of an implementation into shares, and to perform the computations on those shares only. Theoretically, masking is expected to increase the security of the implementation exponentially in the number of shares [52,26,27], with quadratic (area/time and randomness) overheads (see [36,34] for recent examples in hardware and software, respectively).

Despite these theoretical promises (and besides their important performance overheads), the deployment of secure masked implementations is usually slowed down (or sometimes even annihilated) by two types of engineering challenges.

On the one hand, masking leads to an exponential security increase only under strong independence and noise conditions that are non-trivial to ensure.

As far as the independence condition is concerned, three physical defaults are currently mentioned in the open literature. The first one is glitches (i.e. transient computations) that can recombine the shares manipulated by a combinatorial circuit (and typically happen in hardware) [44,45]. The second one is transitions in memories, registers or other (hardware or software) components, possibly leading to shares recombinations as well [23,1]. The third one is couplings, which directly re-combine the shares' leakages due to physical proximity [20,21,42]. These physical defaults lead to contrasted consequences. Glitches and transitions can be captured by abstract models such as Ishai, Sahai and Wagner (ISW)'s probing model [39]. For example, imposing a non-completeness property to the shared functions' executions prevents shares' recombinations due to glitches [50], which can in turn be analyzed formally thanks to a variation of the ISW model with extended probes [9,30]. Similar techniques apply to transition-based leakages. So such defaults can be theoretically mitigated, yet in both cases with additional constraints that usually imply additional performance overheads – the limitation of which has been a very active research line over the last years (see, e.g. [49,22,35]). By contrast, couplings are hard to analyze with mathematical abstractions. So the anticipation (by designers) of the security reductions they imply remains prospective for now, typically leading to an undesirable risk factor to be taken into account in security assessments.

As far as the noise condition is concerned, the main issue is the evaluation of advanced (multivariate, horizontal) attacks taking advantage of multiple manipulations of the shares [5,37]. In this case as well, strong theoretical guarantees (i.e. logarithmic or even constant noise rates) can be obtained, but again at the cost of significant performance overheads [16].

On the other hand, assessing the security of a masked implementation is a time-consuming and technically-challenging process, as witnessed by the multi-model approach introduced in [40]. In its most complete form, such an evaluation typically holds in three stages. One first evaluates the abstract (probing) security of an implementation. But directly testing probing security of full circuits scales badly with the number of shares [2] and usually requires to rely on non-trivial compositional reasoning [3]. Next, quantitative security evaluations must bound the "security order" of the implementations, which corresponds to the smallest key-dependent statistical moment of the leakage distributions (and is needed to validate the independence condition) [54]. But validating this condition rapidly becomes expensive as the number of shares increases [56]. Eventually, the noise level of the implementations must be estimated with metrics such as the Signal-to-Noise Ratio (SNR) or the Mutual Information (MI) [27], and the noise reductions due to the aforementioned advanced (multivariate, horizontal) attacks where multiple time samples are exploited must be taken into account.

Based on this state-of-the-art, our goal in this paper is to improve masked implementations, not in the usual sense of improving the security vs. performance tradeoff, but in the sense of making them significantly easier to implement securely and to evaluate. For this purpose, we investigate the gains that can be obtained in this direction by using a (almost) key-homomorphic re-keying scheme. These are easy-to-mask since they (almost) satisfy the property that $\bigoplus_i f(k_i, \cdot)$ is equal to $f(\bigoplus_i k_i, \cdot)$. Operations on the key can therefore be performed by applying the function to each share independently and then summing the outputs. This brings several advantages to masked implementations that we next denote as *simplicity* and that we detail as:

1. Key-homomorphic primitives are trivial to analyze from the probing security viewpoint, and they can leverage simple and cheap refreshing schemes (with linear overheads), discussed in [4] (Section 8.2).
2. Key homomorphic primitives mitigate the risks of physical defaults leading to shares re-combination. They manipulate shares independently to prevent glitches and transitions. We additionally limit the risks of couplings by parallelizing the computations for each share but not over different shares.
3. Our selected (almost) key-homomorphic re-keying scheme provides inherently a good security against horizontal attacks (e.g. compared to a masked AES implementations) since it manipulates each key bit minimally.
4. The evaluation of the resulting implementations is scalable. Increasing the number of shares for a block cipher implementation usually benefits from order-specific optimizations (e.g. for the randomness) and implies some re-design (i.e. the internal components must be re-synthesized). For our key-homomorphic architecture, increasing the number of shares boils down to re-using exactly the same component (avoiding the repetition of time-consuming and technically-challenging evaluations).

Concretely, we focus on a proposal of re-keying scheme based on Learning With Rounding (LWR) assumption, proposed at Crypto 2016 [29]. It is based on an inner product with vectors having elements in $\mathbb{Z}_{2^q}$ (we consider the case $q = 32$), which leads to very efficient computations and is argued to provide a pseudo random function (PRF) for random public inputs and so makes it a weak PRF (wPRF). We strengthen it by hashing its public input in order to obtain a PRF (in the random oracle model), as suggested in [10]. This selection of a cryptographically robust re-keying function is motivated by the observation that simpler proposals of re-keying schemes such as [47,25] may impose additional constraints on the noise level in order to prevent attacks targeting directly the (leakage of the) fresh keys [8,7,38].

As a result, the main potential drawback of such a masked computation is its performance overheads (e.g.

due to a large key size of 128×32 bits). In this respect, our main contribution is to provide a comprehensive performance and security assessment of a LWR-based re-keying scheme on a recent field-programmable gate array (FPGA) platform. The latter allows us to weight the cost of simplicity and the aforementioned gains it provides, both in terms of risks reduction and in terms of design/evaluation efforts. Precisely, we first compare the cost of this re-keying with the open-source Domain-Oriented Masking (DOM) architecture proposed for the AES in [36][1], which allows us to exhibit very different tradeoffs: the DOM overheads are mostly in circuit size for a constant time, while the ones of our LWR-based re-keying are mostly in cycles. We then analyze their respective security in a quantitative manner (previous works were limited to the assessment of the security order), taking multivariate & horizontal attacks into account and leveraging recent tools to bound physical information leakages [12]. Considering the global performance vs. security tradeoff, we conclude that the cost of simplicity is affordable, and the performances it provides can compete with state-of-the-art block ciphers when high security is required (with pros and cons that may lead to some preferences in specific contexts).

The field of side-channel resistant PRFs has already been investigated as an example with Lapin [32] or Spring [11]. However, they do not enjoy as much simplicity as LWR. Spring is not (almost) key-homomorphic making composition non-trivial. Its most promising FPGA implementation [11] is only partially masked. Lapin is not deterministic since it requires addition with Bernoulli noise. In order to mask the complete circuit, the latter needs to be protected as well which appears to be costly. The difficulty of this noise generation motivates the choice of LWR for masked designs since it is deterministic.

2 Background

In this section, we first highlight the notations used in the rest of the paper. Second, the LWR-based re-keying scheme is recalled. Third, tools used for security evaluations are explained (i.e. leakage detection and certification).

[1] While other *masking friendly* block ciphers could have been investigated, this open-source AES implementation is representative of the state-of-the-art protected designs. This allows us to draw qualitative tradeoffs between the LWR-rekeying scheme and a standard substitution permutation network.

2.1 Notations

The set $\mathbb{Z}_{2^q}$ corresponds to the set of integers modulo 2^q. Vectors of size m of such elements are denoted as $\boldsymbol{Z}_{2^q}^m$. The scalar product between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is written as $< \boldsymbol{x}, \boldsymbol{y} >$. The rounding operation is expressed as $\lfloor x \rfloor_{2^p}$ and is the result of the division of x by 2^p. In binary representation and for an $x \in \mathbb{Z}_{2^q}$, this is implemented by keeping its $q-p$ most significant bits. Similarly, the modulo operation denoted as $x \mod 2^p$ is the rest of the division. In binary representation and for an $x \in \mathbb{Z}_{2^q}$, it corresponds to keeping the p least significant bits of x. The notation $x \leftarrow \mathbb{Z}_{2^q}$ denotes that x is uniformly sampled for the set $\mathbb{Z}_{2^q}$. The d shares of a variable (or vector) x are individually denoted as x_i. The collection $(x)_d$ is for the d shares of x in a masking scheme. The notation $\boldsymbol{x}^{i:j}$ denotes the elements of the vector $\boldsymbol{x}$ from index i to $j-1$.

2.2 Offset Learning with Rounding re-keying

In the following, we explain the LWR-based re-keying scheme from [29]. It generates ephemeral session keys sk based on a secret master key $\boldsymbol{msk}$ and a public random vector $\boldsymbol{R}$. In this setting, the master key is represented with d additive shares $(\boldsymbol{msk})_d$. The secure re-keying scheme is then based on the three different steps described below:

– $\mathsf{Gen}(1^\lambda)$: initializes the master key $\boldsymbol{msk}$ and its shares. First, $\boldsymbol{msk}$ is sampled such that $\boldsymbol{msk} \leftarrow \boldsymbol{Z}_{2^q}^m$. Then, the d shares $(\boldsymbol{msk})_d$ are sampled uniformly from all the possibilities that fulfill the equation $\boldsymbol{msk} = \sum_{i=1}^d \boldsymbol{msk}_i$. More precisely, $d-1$ shares are picked up from a uniform distribution while the last one is computed as $\boldsymbol{msk}_d = \boldsymbol{msk} - \sum_{i=1}^{d-1} \boldsymbol{msk}_i$.
– $\mathsf{GenSk}((\boldsymbol{msk})_d, \boldsymbol{R})$: generates a session key sk of p' bits from a uniformly distributed vector $\boldsymbol{R} \leftarrow \boldsymbol{Z}_{2^q}^m$. First, for each shares of the master key $\boldsymbol{msk}_i$, a share of session key is obtained with a scalar product and a rounding

$$sk_i = \lfloor < \boldsymbol{msk}_i, \boldsymbol{R} > \rfloor_{2^p}. \tag{1}$$

Second, the session key sk and correction factor v are obtained by summing the $(sk)_i$ such that

$$sk := \left\lfloor \sum_{i=1}^d sk_i \right\rfloor_{2^{p'}} \tag{2}$$

and

$$v := \sum_{i=1}^d sk_i \mod 2^{p-p'}. \tag{3}$$

The session key sk corresponds to the p' upper bits of the sum while v corresponds to its $p-p'$ lower bits. Finally, the key shares must be refreshed such that $(\boldsymbol{msk})'_d \leftarrow \mathsf{Refresh}((\boldsymbol{msk})_d)$ and we use the optimal (linear) refreshing from [4] for this purpose (i.e. we add a share of zero).

– $\mathsf{corrSk}(\boldsymbol{msk}, \boldsymbol{R}, v)$: allows to retrieve the session key sk with the knowledge of the $\boldsymbol{msk}$, the public $\boldsymbol{R}$ and the correction factor v by correcting the errors due to the rounding operation. First, the scalar product is computed with $y := \lfloor < \boldsymbol{msk}, \boldsymbol{R} > \rfloor_{2^p}$, then the session key is retrieved thanks to

$$sk := y + (v - y \mod 2^{p-p'}) \mod 2^p. \tag{4}$$

The above parameters must fulfill the condition $q > p > p'$, $2^{p-p'} > d$ to allow to correct the errors in $\mathsf{corrSk}(\cdot)$.

In order to turn such a wPRF into a PRF, the uniformly distributed vector $\boldsymbol{R} \leftarrow \boldsymbol{Z}_q^m$ is generated from a call to a random oracle O, which we instantiate with a hash function such that $\boldsymbol{R} \leftarrow \mathsf{H}(y)$, where y can be chosen by the adversary [10].

We next focus on the $\mathsf{GenSk}(\cdot)$ algorithm since it has to be executed online, potentially on an exposed embedded system, and therefore has to be securely masked. By contrast, $\mathsf{Gen}(\cdot)$ is run only once in a personalization phase.[2]

2.3 Security evaluation tools

The goal of our analysis is to approach the worst-case security level of our target implementations. Such an analysis attempts to give a lower-bound on the data complexity of any side-channel attack [55]. This covers any differential power analysis (DPA) such as (higher-order) correlation power analysis, template attacks, deep-learning attacks as well as horizontal attacks. For this purpose, we follow the approach outlined in the introduction which is based on masking proofs and information theoretic arguments instead of attack based evaluations. We first leverage the fact that masked key-homomorphic primitives do not suffer from composability issues. We then assess the independence condition thanks to leakage detection tests, and we finally bound the security level (i.e. the attack complexity) thanks to an information theoretic analysis. Precisely, in [27], it is shown that the worst-case complexity of any side-channel attack against a masked implementation can be bounded thanks to the MI between the leakage L and the secret K, which can itself be bounded by the product of the MI on each share:

$$\mathrm{MI}(K; L) \leq \prod_{i=1}^{d} \mathrm{MI}(K_i; L). \tag{5}$$

As a result, and given that the independence condition is fulfilled, we can exploit this shortcut and bound the security by computing the (much easier-to-estimate) MI per share. The bound then highlights the noise amplification mechanism that masking ideally provides. We additionally exploit recent results in leakage certification to make sure that our bounds are independent of model errors that could bias our conclusions [12].

Independence Assessment. The Test Vector Leakage Assessment (TVLA) [6,33] is a popular tool in security evaluation labs, which is generally used as the first step in validating the resistance of a device to side-channel attacks. This methodology is based on Welch's t-test [57] which is a standard statistical test to detect difference in means between two populations. The leakage assessment is performed in two steps which can be efficiently implemented [54]. First, the device is fed with two different sets of inputs $\mathcal{I}_1$ and $\mathcal{I}_2$ and the corresponding leakages $\mathcal{L}_1$ and $\mathcal{L}_2$ are collected. [3]. Second, a test statistic t is computed as

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_2}{\sqrt{\frac{\hat{\sigma}_1^2}{N_1} + \frac{\hat{\sigma}_2^2}{N_2}}}, \tag{6}$$

where $\hat{\mu}_i$ and $\hat{\sigma}_i^2$ respectively represent the estimated mean and variance from N_i samples. If $|t|$ exceeds a given threshold, the test showed difference in means and the chip is considered as insecure. The selection of the threshold depends on the statistical confidence required and the trace length [24,13] but is commonly assumed to be 4.5. The proposal in [54] additionally describes how to extend this methodology to multivariate and higher-order detections.

Finally, the link between (higher-order) detections and the independence assumption is for example explained in [56]: in a masked implementation with claimed security of order d, all the statistical moments smaller than d of the $\mathcal{L}_1$ and $\mathcal{L}_2$ sets should be equal [4].

Noise estimation. The second step of our evaluations is to estimate the MI between a single share and the corresponding leakages. To do so, we use the recent tools from [12] and also proceed in two steps.

[2] Securing $\mathsf{corrSk}(\cdot)$ which is executed by the decryption party can be done with similar means, by increasing the amount of correction information by one bit.

[3] Typically, the first set correspond to fixed input. The second set can be either chosen to correspond to random inputs or to another fixed input [28]. We detail our selection of inputs for LWR in subsection 5.2

First, we exploit the generic upper bound that this paper provides, and which allows bounding the (unknown) MI metric independent of the leakage model used by the adversary. Since this bound converges slowly for larger number of dimensions, we only compute it for univariate attacks (i.e. for all the samples of our traces independently) and then compare it to a similar bound corresponding to a Gaussian adversary. Given that we observe no significant discrepancies between the non-parametric and Gaussian bounds, we then generalize the Gaussian analysis to a (close to worst-case) highly multivariate adversary for which [12] provides an efficient bound. It exploits the differential entropy $\mathrm{H}(X)$ of a multivariate Gaussian distribution X defined as

$$\mathrm{gH}(X) \leq \frac{\log(\det(2\pi e \Sigma))}{\log(2)}, \tag{7}$$

with Σ the covariance matrix and $\det(\cdot)$ the matrix determinant. Then, it uses the standard expression for mutual information

$$\mathrm{MI}(K_i, L) = \mathrm{gH}(\hat{\mathsf{L}}(K_i)) + \mathrm{gH}(L) - \mathrm{gH}(\hat{\mathsf{L}}(K_i); L), \tag{8}$$

with $\hat{\mathsf{L}}(K_i)$ the noise-free part of the leakage model (i.e. the multivariate Gaussian templates [18]) estimated by the adversary/evaluator. In other words, this model corresponds to the mean of L when a given value is manipulated. It can either be computed by performing direct averaging on L, or by performing a linear regression [53], which is more efficient for target keys with large bitsizes. Evaluating Equation (8) can be done by computing its terms independently, and by re-using the same traces as used to build the model. This (overfitting) method is proven to provide an upper bound on the MI, denoted as the *Hypothetical Information* (HI), here specialized under the Gaussian assumption and denoted as gHI. The tightness of the bound depends on the number of measurements available (the more measurements, the tighter the bound).

Putting all together, once the security order of an implementation is shown to be equal to d based on the Welch's t-test, the estimation of $\mathrm{MI}(K_i, L)$ can be plugged in Equation (5), which leads to a simple lower bound on the data complexity of the side-channel attack of $\frac{c}{\mathrm{MI}(K,L)}$, with c a small constant dependent on the target success rate and target bitsize (e.g., $c \approx \mathrm{H}(K)$ corresponds to a success rate of approximately 80% [19]).

3 Hardware (FPGA) architecture

In this section, we describe our hardware architecture for an LWR-based re-keying implemented on a FPGA platform. We first recall our design goals and provide an overview of the functional blocks to be implemented, and then present their design together with security considerations.

3.1 Design goals and overview

In terms of implementation security, the critical block of the investigated LWR-based re-keying is the $\mathsf{GenSk}(\cdot)$ algorithm. Since it manipulates the long-term master key, it is a natural target for a DPA and therefore has to be protected against such attacks. In the context of masking, it means that our design has to be engineered in order to reach two main goals: first, the architecture has to ensure the independence of the shares' leakages; second, it has to limit the opportunities of multivariate/horizontal attacks, and ideally to enable an easy estimation of the MI metric (per share). As mentioned in introduction, these goals are quite directly achieved by a natural implementation of a key homomorphic primitive. The only constraint is that our architecture should never process the shares in parallel, in order to lower the risks of couplings [20, 21, 42]. Based on this simple guideline, the same hardware can be recycled to deal with all the shares, also making the MI metric estimation equivalent for all of them.

The sequence of operations that are performed by the re-keying core are contained in the pseudo-code of Algorithm 1. It shows how the equations from subsection 2.2 are computed. More precisely, all the session key shares sk_i are processed serially: they are obtained by computing the scalar product between the corresponding master key share $\boldsymbol{msk}_i$ and the random vector $\boldsymbol{R}$. Once an sk_i is computed, it is rounded to 2^p and accumulated in x. The returned values are the most significant bits of x that correspond to the session key sk, and the correction factor v. In order to gain

Algorithm 1 Session key generation $\mathsf{GenSk}(\cdot)$.

Input: $(\boldsymbol{msk})_d$ with $\boldsymbol{msk} = \sum_{i=1}^{d} \boldsymbol{msk}_i$ and $\boldsymbol{R} \leftarrow \mathbb{Z}_{\boldsymbol{2^q}}^{\boldsymbol{m}}$.
Output: Session key sk and correction factor v.

```
$x = 0$;
for $i \in [1, \ldots, d]$ do
    $sk_i = 0$;
    for $j \in [0, m/P - 1]$ do
        $a = < \boldsymbol{msk}_i^{j*P:(j+1)*P}, \boldsymbol{R}^{j*P:(j+1)*P} >$;    ▷
$\quad$Addition Tree
        $sk_i = sk_i + a$;    ▷ Scalar product
    $x = x + \lfloor sk_i \rceil_{2^p}$;    ▷ Sum
return $\lfloor x \rceil_{2^{p'}}$ , $x \mod 2^{p-p'}$.
```

in efficiency, a level of parallelism is allowed in our im-

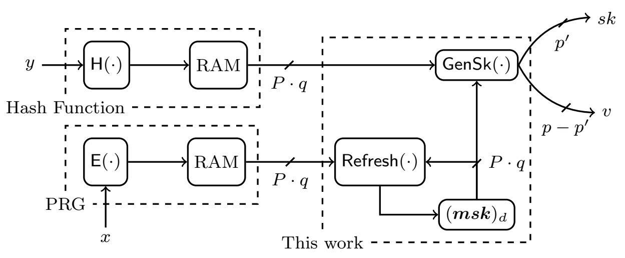


Fig. 1: Re-keying architecture.

plementations. As per our goals, we cannot parallelize the processing of the shares, but we can parallelize the multiplications of several elements within the vectors $\boldsymbol{msk}_i$ and $\boldsymbol{R}$. In the rest of the paper, P denotes a parallelism coefficient, which corresponds to the number of q-bit elements processed per cycle. One may note that a software implementation would directly apply this algorithm with $P = 1$. Since additions and multiplications over $\mathbb{Z}_{2^q}$ are available in most of modern processors [58], the software code is straightforward.

The previously exposed pseudo-code can be implemented based on the block diagram of Figure 1 with three main components. The first component is $\mathsf{GenSk}(\cdot)$ and corresponds to the circuit implementing the scalar products and the accumulations. It takes as first input the master key shares $\boldsymbol{msk_i}$, which are stored in memories and connected to the $\mathsf{Refresh}(\cdot)$ module. The latter is responsible for refreshing the shares once they are used by $\mathsf{GenSk}(\cdot)$. In order to perform the refresh, the randomness is sampled from a Pseudo-Random Generator (PRG). All these modules are manipulating secret information and therefore need to be carefully implemented to avoid side-channel attacks. In the following, we will describe in details how they can be synthesized.

The second input of $\mathsf{GenSk}(\cdot)$ is the random public vector $\boldsymbol{R}$. In order to turn the LWR-based wPRF of [29] into a PRF (in the random oracle model), we simply hash the input y. Since the hash function manipulates public data, it does not require particular care from the side-channel resistance viewpoint. We use the SHAKE-256 hash function for this purpose, which outputs vectors of arbitrary sizes. This module is based on the publicly available code of Keccak (we selected the high-speed core from `https://keccak.team/hardware.html`).

3.2 Scalar product computations

The $\mathsf{GenSk}(\cdot)$ architecture that is in charge of computing the scalar products is depicted in Figure 2, where each of the pipeline stages corresponds to a line in the previous pseudo-code of Algorithm 1.

The leftmost one contains the addition tree which performs the partial scalar product on (sub)vectors of size P. It computes the element-wise multiplications between $\boldsymbol{msk}_i$ and $\boldsymbol{R}$ and stores the results in registers. These elements are then added together within a binary addition tree of depth $\log_2(P)$.

In the next stage, the scalar product $< \boldsymbol{msk_i}, \boldsymbol{R} >$ is obtained by accumulating the m/P outputs of the addition tree.

The last stage is used to sum over all the sk_i's to build the correction factor v and the session key sk which are stored in a register.

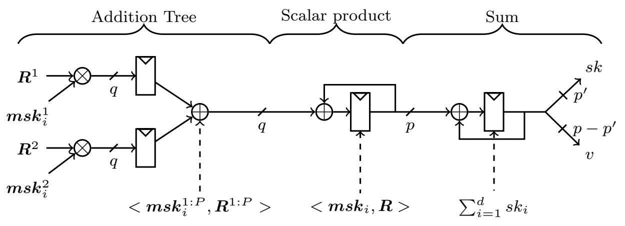


Fig. 2: $\mathsf{GenSk}(\cdot)$ with $P = 2$.

As illustrated on the figure, the parameter P allows reducing the latency of this module. Computing P multiplications in parallel divides the number of cycles to compute a scalar product by a factor P. This requires to access P elements of $\boldsymbol{msk}_i$ and $\boldsymbol{R}$ in parallel, which increases the width of the memory used (see the next subsection). Besides performance gains, computing P multiplications generates algorithmic noise within the design which is beneficial from the security viewpoint. More precisely, for an adversary mounting a b-bit divide-and-conquer side-channel attack, the $P-1$ other multiplications will act as an additive noise source. In order to mitigate this noise increase, the adversary can try to target larger parts of the secret (i.e. increase b). But already for $P = 1$, targeting $b = q = 32$ bits at once is computationally intensive [46].

3.3 Master key storage

In order to protect the secret key against side-channel attacks, it is crucial to manipulate its shares carefully. Precisely, the architecture must store the d shares $\boldsymbol{msk}_i \in \mathbb{Z}_{2^q}^m$ of the master key. Each of these are composed of m words of q bits leading to a total memory of at least $m \cdot q \cdot d$ bits (i.e., $128 \cdot 32 \cdot d$ bits for the parameters used in the next sections). The designer then has to choose how to allocate resources, which in our case has to be driven by side-channel security. As depicted in Figure 2,

P words of $\boldsymbol{msk}_i$ must be accessed during a single cycle in order to enable parallel computations. Therefore, the memories used in the architecture are of width $P \cdot q$, as shown in Figure 3. The key shares are accessed with two inputs: they correspond to the indices i and j of Algorithm 1, and give access to $\boldsymbol{msk}_i^{j*P:(j+1)*P}$. Once available, these shares are directly fed into the $\mathsf{GenSk}(\cdot)$ module.

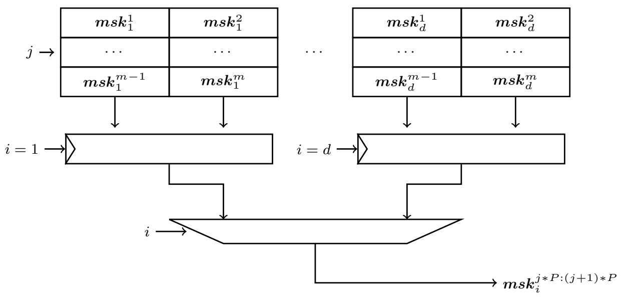


Fig. 3: $(\boldsymbol{msk})_d$ storage for $P = 2$.

We add two constraints to facilitate the independence condition:

First, we store the shares in distinct memory blocks. By doing so, we avoid location proximity between them. Since the designer does not always have full control on the memory blocks (especially on FPGA platforms), this ensures that no physical recombination of the shares can happen due to physical defaults within a single memory block.

Next, by processing the shares serially, we prevent that they are recombined within a single pipeline stage. However, since d different memory blocks are used to feed the $\mathsf{GenSk}(\cdot)$ block, the logic for selecting the correct memory output should be designed carefully. A simple solution would be to set a multiplexer, but glitches could occur within that logic. To avoid them, the inputs of the multiplexer should not be related to key shares excepted for the one that is accessed. For this purpose, we insert registers at the output of each of the d memories that are reset if the memory i was not accessed during the last cycle. Sequentially, the operations performed are the following. On the first cycle, the memory is accessed to obtain the j-th row for the share i, namely $\boldsymbol{msk}_i^{j*P:(j+1)*P}$. On the second cycle, the obtained data is stored in the corresponding register that was containing either zeros either data from the (j-1)-th row that is uncorrelated with the j-th row. On the last cycle, the multiplexer fetches data from the register corresponding to share i. Note that this approach adds one cycle of latency on the memory block. Since all the computation is sequential, this does not harm the performances. A similar approach is used for the write port of the memories.

3.4 Refreshing of the shares

After each inner product computation, the master key has to be refreshed. Such a refreshing scheme takes as input the shares $(\boldsymbol{msk})_d$ of a secret and outputs a new set of shares $(\boldsymbol{msk'})_d$. The later is sampled uniformly among all the possibilities fulfilling the equation $\boldsymbol{msk} = \sum_{i=1}^{d} \boldsymbol{msk}'_d$. As pointed out in [4], for key-homomorphic primitives, the secret can be refreshed in a very simple manner. More precisely, in this context a linear refresh is sufficient. The refreshing scheme is shown in Algorithm 2. It takes as input the shares $(\boldsymbol{msk})_d$ as well as a matrix $\boldsymbol{r}$ of size $d \times m$ sampled from the uniform distribution. Each share msk_i^j within $(\boldsymbol{msk})_d$ is replaced by its sum with the two random values r_i^j and $-r_{i+1}^j$. In a vectorized form, this corresponds to $\boldsymbol{msk}^j + \boldsymbol{r}^j - rot(\boldsymbol{r}^j)$ where $rot(\cdot)$ denotes the rotation on the vector elements. After the refresh, the share $msk_i'^j$ is a new random value with uniform distribution. For correctness, this scheme also ensures that the sum of the $\boldsymbol{msk}'_i$'s is equal to $\boldsymbol{msk}$, since the elements of $\boldsymbol{r}$ are simplified with their negative correspondent.

Algorithm 2 Linear refreshing $\mathsf{Refresh}(\cdot)$.

Input: $(\boldsymbol{msk})_d$ with $\boldsymbol{msk} = \sum_{i=1}^{d} \boldsymbol{msk}_i$ and $\boldsymbol{r} \leftarrow \mathbb{Z}_{\boldsymbol{2^q}}^{\boldsymbol{m \times d}}$.
Output: $(\boldsymbol{msk'})_d \leftarrow \mathsf{Refresh}((\boldsymbol{msk})_d)$ with $\boldsymbol{msk} = \sum_{i=1}^{d} \boldsymbol{msk}'_i$.

```
for $i \in [1, \dots, d]$ do
    for $j \in [1, \dots, m]$ do
        $msk_i'^j \leftarrow msk_i^j + r_i^j - r_{i+1}^j$;
return $(\boldsymbol{msk})_d$
```

This $\mathsf{Refresh}(\cdot)$ operation has to be performed every time a share is accessed. If not, an adversary could observe multiple times the leakage related to each share of the master key. This would allow him to reduce the noise on its observations of the shares and as a result, the security of the device. Concretely, every time a share is read from the memory of Figure 3, it is directly refreshed and written back in memory. The parallel computation of these two operations is straightforward since the shares have to be accessed consecutively in both $\mathsf{GenSk}(\cdot)$ (i.e. Algorithm 1) and in $\mathsf{Refresh}(\cdot)$ (i.e. Algorithm 2).

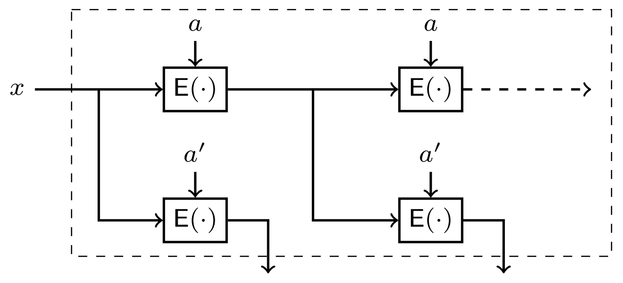


Fig. 4: Leakage-Resilient Pseudo-Random Generator.

3.5 Randomness generation

One of the inputs of the $\mathsf{Refresh}(\cdot)$ block is a random matrix. The exact requirements for this randomness are unfortunately an under-studied problem in the masking literature. In order to obtain a good overview of the performances of our re-keying scheme, we considered two solutions for this purpose.

The first (expensive and conservative) solution is to generate the randomness with a Leakage-Resilient Pseudo-Random Generator (LR-PRG) such as [51], illustrated in Figure 4. Such a construction iterates a block cipher $\mathsf{E}(\cdot)$ in a forward-secure mode of operation. More precisely, the PRG takes as input a secret seed x, which is used as an encryption key in $\mathsf{E}(\cdot)$ to encrypt a public constant a'. The result is then outputted by the PRG as a fresh random value. The seed x is also used to encrypt a second public constant a, producing a ciphertext used as a new ephemeral key in the next PRG iteration. Doing so, each ephemeral key is used only twice. Since this process is typically expensive (and our masked implementations require a lot of randomness), we will consider a setting where the required randomness is computed on-the-fly when using such a conservative solution (therefore leading to overheads in cycles).

Alternatively, a much cheaper (and much less conservative) solution is to use a simple LFSR of the appropriate size. In this case, the randomness generation is so fast that it can be produced on-the-fly, leading to no cycles overheads, but an increasing area when the required randomness per cycle increases (which typically happens when increasing d).

Overall, once the choice to avoid manipulating shares in close time and space location has been made, the design space remains small since the LWR itself relies only on scalar products. The main design choices left are the selection of the hash function and the PRG. For the former, we selected SHAKE because it is well studied. It is not lightweight which makes it a worst-case for our cost metric as discussed in the next section. The impact of the PRG is discussed in the next section.

4 Performance and cost

The side-channel protection provided by masking comes with performance overheads. In the following, this additional cost is discussed in terms of time and area for two fundamentally different architectures. On one hand, the LWR-based re-keying uses constant computational resources independently of the number of shares. However, its latency grows in $\mathcal{O}(d\log_2(d))$ (where the log factor comes from the need of more error correction when d increases), which can be mitigated thanks to the parallelism factor P. Its memory requirements are linear with d. On the other hand, the AES implementation protected with Domain Oriented Masking (DOM) proposed in [36] has a constant time for any d but its area grows in $\mathcal{O}(d^2)$. We next denote that implementation as AES-DOM.

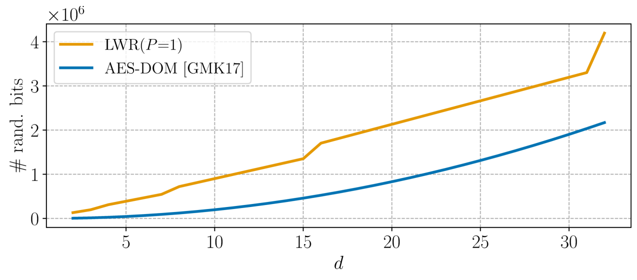
We evaluated performances on an FPGA (`Xilinx Kintex-7`) platform, and considered the generation of 128 bits of session key. In the case of the AES-DOM architecture, this corresponds to a single encryption. For the LWR-based re-keying, the selected parameters are the same as the ones proposed at Crypto 2016: $q = 32$, $p = 10$ and $m = 128$ [29]. Our evaluations report two metrics called *cost* and *cycles*.[4] The cost metric is the sum of the number of look-up-tables (LUTs) and registers used by the implementation. This cost metric does not include the Block RAM's used to store the key in the LWR-based scheme while these are not used by AES-DOM. For fairness, the registers used to store the key with in AES-DOM implementation are so not taken into account.[5] The utilization of Block RAM and DSP of LWR are detailed in Figure 8b and subsection 4.2. The cycles metric is simply the number of clock cycles required to perform a given operation.

4.1 Randomness requirements & PRG impact

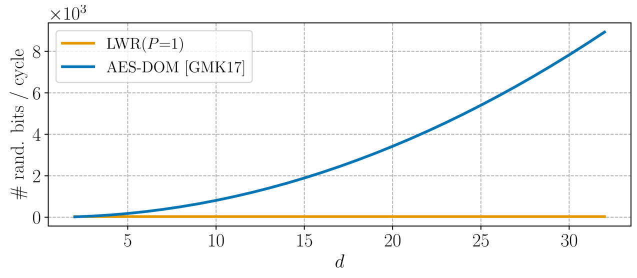
We start by evaluating the amount of randomness required by both architectures and consider the total amount and the amount per cycle, which sets constraints on the PRG. Based on these, the impact on the global architecture of the two PRGs in subsection 3.5 can be assessed.

[4] All the resource utilization results reported are obtained with `Xilinx ISE 14.7`.

[5] These have a limited impact on the metric for large d. Indeed, they grow linearly with d while the rest of the circuit is quadratic.



(a) Total number of rand. bits.



(b) Number of rand. bits per cycle.

Fig. 5: Randomness requirement depending on the masking order d.

Randomness requirements. In the case of the LWR-based re-keying, the total number of random bits needed to generate a 128-bits session key is related to the number of calls to $\mathsf{GenSk}(\cdot)$. Indeed for each session key sk of p'-bits, the master key has to be refreshed thanks to $\mathsf{Refresh}(\cdot)$ which takes as input a total of $m \cdot q \cdot d$ fresh random bits. Additionally, an increased number of shares reduces the size of the session keys because of the condition $2^{p-p'} > d$ imposed for the error correction $\mathsf{corrSk}(\cdot)$. As a result, the number of random bits is growing in $\mathcal{O}(d \log_2(d))$ and is depicted in Figure 5a. The steps in the curve correspond to additional calls to $\mathsf{GenSk}(\cdot)$. In our architecture, only $q \cdot P$ bits of the master key are processed in parallel, which keeps the number of random bits to produce per cycle at a constant level (see Figure 5b). We only report the $P = 1$ case; the larger P's are obtained by direct multiplication.

The situation of the AES-DOM architecture is quite different: in this case, both the total amount of randomness and its requirements per cycle grow in $\mathcal{O}(d^2)$ (due to a constant time architecture). For the total amount of randomness, and even for $d = 32$ shares, this is not enough to compensate the large constant overheads of the LWR-based re-keying. Hence, the most striking difference between the two schemes is in the amount of randomness needed per cycle, which is not due to the algorithmic differences but to architectural choices.

PRG impact. As already mentioned, exact requirements for the quality of the randomness in masking are currently unknown. So we study the impact of two extreme choices. First, a LR-PRG which corresponds to a conservative and expensive solution. Next a simpler and cheaper LFSR of the appropriate size.

In case the LR-PRG is used, two block-cipher instances are executed per 128 bits of randomness.[6] The number of cycles required to output a given number of random bits with such a PRG is shown in Figure 6a. For example, approximately 10^7 cycles are needed to generate all the randomness required by a LWR-based re-keying with 32 shares. This number can be reduced by duplicating the number of LR-PRGs used (i.e. by using P LR-PRGs in parallel). This linearly decreases the latency and increases the number of random bits delivered per cycle with a proportionally increased cost. Such a PRG can be used on-the-fly for both the AES-DOM and the LWR-based re-keying, in which case it typically dominates the latency overheads of the system when d increases. In the case of our re-keying-based architecture, one could also consider using the LR-PRG in a constant-time manner, since the amount of random bits per cycle is constant in d. By contrast, it is unrealistic in the AES-DOM case (as per the vertical lines in Figure 6b, which show the randomness requirements per cycle for $d = 32$).

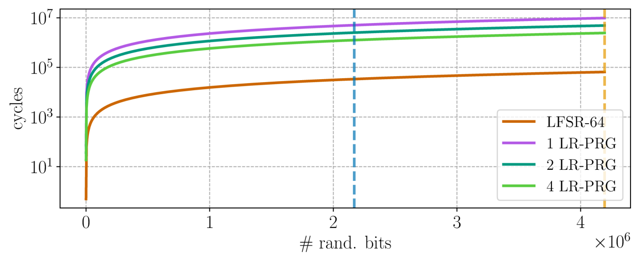
In case the LFSR-based PRG is used, the randomness generation is in general much faster (see Figure 6a), and for a fixed number of random bits per cycle, it is also much cheaper in terms of our cost metric, as depicted in Figure 6b. In the (natural) setting where this randomness is generated on-the-fly, it causes zero cycles of overheads and only cost increases. Such costs are quadratic in d for the AES-DOM architecture (again due to the constant time feature) and constant in d for the LWR-based re-keying.

Note that the impact of the PRGs used for the shares generation on the security of a masking scheme depends on their security against side-channel attacks with unknown plaintexts / ciphertexts. Attacking a hardware LR-PRG is probably very challenging in this setting, while attacking an LFSR is an easier target due to the simpler algebraic structure. We refer to [14] for an example of such an attack, and insist that we give the LFSR-based evaluations as a non-conservative comparison point (which is used in other papers on masking).

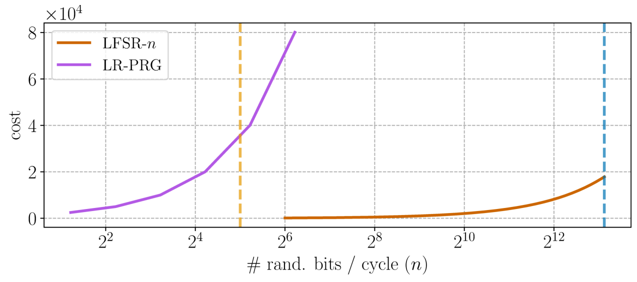
4.2 Resources utilization

In the following, the area and the latency of the re-keying scheme are studied, excluding the randomness

[6] In the following evaluations, it is implemented with two unprotected AES cores delivering 128 bits of randomness every 55 cycles.

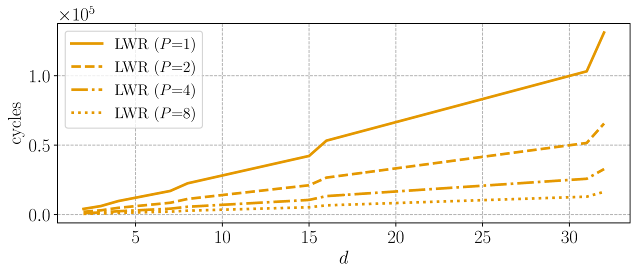


(a) Cycles for a given # of rand. bits.

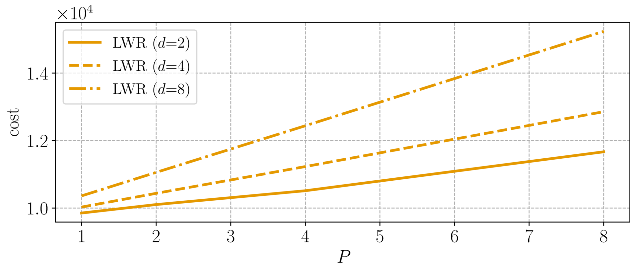


(b) Cost for given # rand. bits per cycle.

Fig. 6: Comparison between two PRGs. Blue vertical lines correspond to the AES-DOM; orange ones to LWR (P=1), both protected with $d = 32$ in Figure 5).



(a) Cycles for a given number of shares d.



(b) Logic utilization for given P.

Fig. 7: Impact of the parameter P on the LWR-based re-keying performances.

generation for which the additional costs have just been discussed, and comparing it with the AES-DOM architecture. The influence of the main parameters P and d are investigated in a systematic manner.

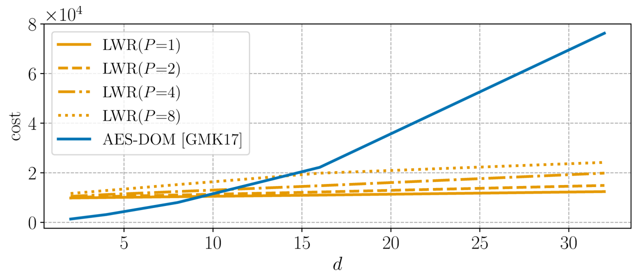
Parallelism. We start with an analysis of the parameter P which is specific to the LWR-based re-keying architecture, based on our two main metrics. First, in Figure 7a, the number of cycles needed to generate a fresh session key is reported. It shows that the latency decreases thanks to the P parallel multiplications performed for a single scalar product. Second, the (larger) resources required to perform these parallel multiplications are illustrated in Figure 7b. This increase is caused by the width of the memories (of Figure 3) which increases to $q \cdot P$, meaning that larger registers have to be inserted at their output. Besides, on the FPGA platform we consider, the multiplications can be implemented within optimized cells for arithmetic operations called DSPs. A single 32-bit multiplication uses four of those. Because of the linearly increasing number of independent multiplication with P, the number of DSPs increases accordingly. Eventually, due to the longer depth of the addition stage in Figure 2, the maximum clock frequency is reduced from 122[MHz] (for $P = 1$) to 92[MHz] (for $P = 8$).

Number of shares. As previously mentioned, the number of shares d increases the number of clock cycles of the LWR-based re-keying in $\mathcal{O}(d \log_2(d))$. It is shown in Figure 7a. The number of shares also has an influence on the memory and logic elements' utilization. As far as the memory is concerned, all the shares have to be stored, leading to a linear increase of the total memory requirements. Practically, the shares are stored in *Block RAMs* that are fixed-size memory units (of 18[Kb]) available in the FPGA. In order to avoid physical proximity for the shares, we systematically use d memory cells even if they are individually not full (see Figure 8b). So single memory units are used to store sets of $128 \cdot 32$ bits, which corresponds to approximately 4.3% of their total capacity. The parameter d also slightly increases the cost metric of the implementation, as illustrated in Figure 8a, mostly due to registers at the output of the memories.

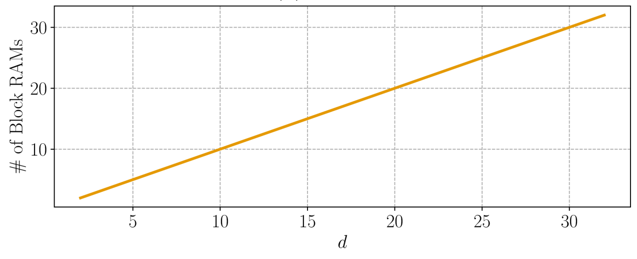
Globally, the impact of d highlights the different implementation choices of the AES-DOM and LWR-based re-keying architectures. The first one has a constant time in d, which comes at the cost of a circuit size increasing in $\mathcal{O}(d^2)$. The second has limited cost overheads (in d) but much higher cycle counts.

4.3 Trade-offs

We further highlight the trade-offs provided by the two types of masked implementations by reporting their cost×cycles metric in Figure 9. We observe that the factor P improves this metric for the LWR-based re-keying if $\mathsf{GenSk}(\cdot)$ is the bottleneck operation, because the marginal additional cost it implies is balanced by a more significantly reduced time complexity. If not

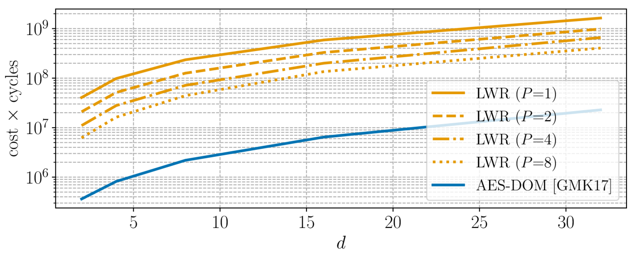


(a) Cost.

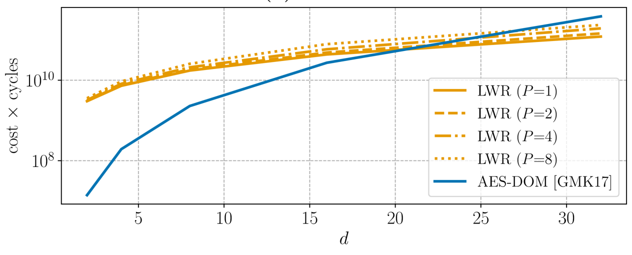


(b) Block RAM utilization.

Fig. 8: Influence of the number of share on the cost of the implementation.



(a) LFSR.



(b) LR-PRG.

Fig. 9: Cost×cycles metric for both LWR and AES-DOM.

(e.g. with an LR-PRG), then it marginally increases the cost×cycles metric due to its slight area overhead (see Figure 7b). More generally, the figure illustrates the impact of the (type of) PRG in the respective cost of these architectures.

If a LFSR generating on-the-fly randomness is used (as in Figure 9a), the LWR re-keying scheme performs worse, independently of the parameter P and security order. In this case, the difference in latency between the AES-DOM and the LWR-based implementation is not compensated by the smaller area of the LWR-based implementation for large d's.

By contrast, if the LR-PRG is used the randomness generation becomes the bottleneck for both implementations. LWR-based re-keying and AES-DOM require (more) similar total amounts of random bits (as shown in Figure 5a). In such a case, for a small number of shares (i.e. $d < 20$), the cost and cycles metrics are smaller in AES-DOM compared to the LWR-based re-keying. By contrast, for large number of shares ($d > 20$), the need in randomness remains smaller for AES-DOM but is compensated by its quadratic cost overheads. Therefore in Figure 9b, the curve for AES-DOM is below the one for LWR at $d = 2$ and above at $d = 32$.

Eventually, adding parallelism for AES-DOM will imply running multiple Sboxes (n) in parallel instead of a single one. This will increase by a factor n the cost of the circuit since it duplicates Sboxes which have a quadratic cost. This will also divide by a factor n the number of clock cycles (in case of an LFSR used as a PRG). Overall, this will lead to a similar *cost* $\times$ *cycles* metric. Performing a quantitative cost analysis of parallelism would require to modify the open-source AES-DOM architecture and is out of scope of this work.

In summary, one can expect that the LWR-based re-keying architecture may become a useful alternative to a standard block cipher implementation when high security levels are required. The next section will show that this conclusion is amplified when security is taken into account.

5 Security evaluation

We now provide a worst-case side-channel security evaluation of our design, based on the bound of [27] and the methodology detailed in subsection 2.3. First, a description of the measurement setup is given together with the design parameters of the evaluated implementation. Second, the independence of the leakage associated to each share is assessed thanks to Welch's t-test [33,54]. Third, the information about a single share available to a multivariate adversary is bounded [12]. Finally, the required number of shares to achieve a targeted attack complexity is extrapolated. These results are compared with the open-source AES-DOM core running under the same conditions.

5.1 Measurement setup and adversarial settings

The next security evaluation assumes an adversary targeting a transition at the output of the multiplier in Fig-

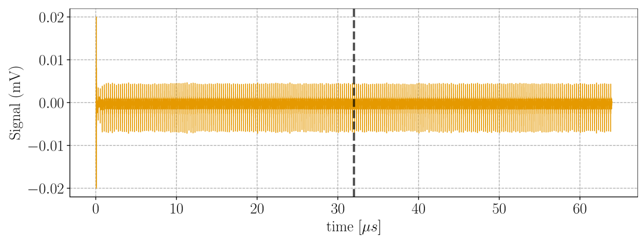


Fig. 10: LWR-based re-keying: mean trace.

ure 2.[7] It corresponds to a non-linear operation mixing the secret shares $\boldsymbol{msk}_i^j$ with a public input $\boldsymbol{R}$, and is therefore a sweet spot for DPA. We considered such a standard divide-and-conquer adversary, with an adversary able to guess the 32 output bits of the multiplier at once.

The studied design is synthesized with a level of parallelism $P = 1$ which is the easiest possible target (larger P values would increase the amount of algorithmic noise). All the inputs required by $\mathsf{GenSk}(\cdot)$ and $\mathsf{Refresh}(\cdot)$ are precomputed. By doing so, the hash function $\mathsf{H}(\cdot)$ and the block cipher $\mathsf{E}(\cdot)$ in the LR-PRG are idle during the scalar product and all the circuit activity is directly related to the targeted secret information.

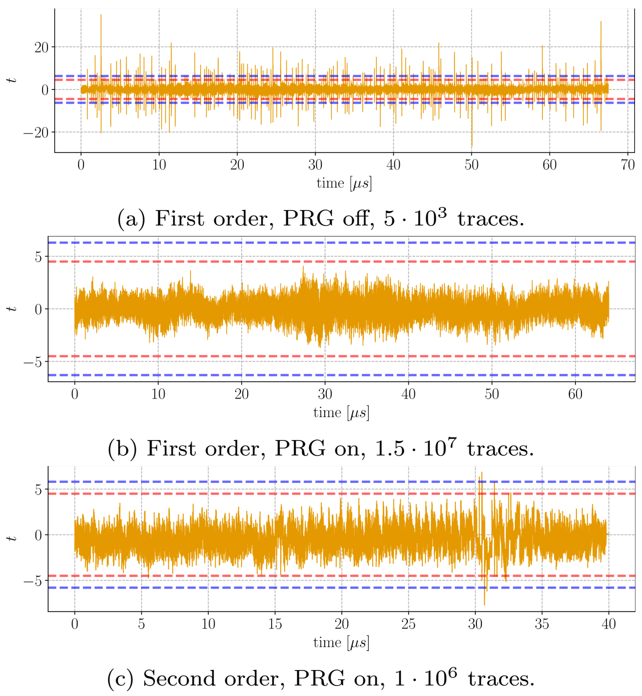
The previously described design was synthesized for a `Xilinx Kintex-7`, running on a dedicated board for side-channel evaluations.[8] Its clock frequency was set to 4[MHz]. The synthesis was performed with the *keep hierarchy* flag avoiding the tool to trim out useful registers (e.g. the ones at the output of the memories in Figure 3). The power signal was measured on a 1[Ohm] resistor placed between the target FPGA and the 1[V] supply voltage. This signal was sampled with a `Picoscope 5244d` at a rate of 500[MSamples/s], with 12-bit resolution. As a result, a leakage trace contains 125 samples per clock cycle.

An averaged trace is shown in Figure 10 for $d = 2$ shares. The left part of the power measurement corresponds to the 128 cycles used to compute sk_1 while the right part is dedicated to sk_2.

5.2 Independence condition

The first step in our worst-case security evaluation is to assess the independence of the leakage of each share. To do so, the TVLA method was used to observe the practical security order of the implementation, which should be equal to the number of shares. For example, in a $d = 2$ case, the mean of the traces should not depend on manipulated data while the variance should. To run the TVLA, two sets of inputs were selected, each of them leading to different intermediate values. The first set is selected with a random key (changing at every encryption) and the second to a fixed key (so we used a "fixed versus random" detection). In both cases, the input of the random oracle is identical and constant.

Our TVLA experiments are presented in Figure 11. These graphs represent the t statistic of Welch's t-test (recalled in Equation (6)) on each of the samples within the traces. Figure 11a shows the results when the PRG is disabled (meaning that the secret key $\boldsymbol{msk}$ is never refreshed). Under these conditions, first-order leakages are detected with $5 \cdot 10^3$ measurements. The PRG is turned on in Figure 11b (meaning that the target is running under nominal conditions). No first-order leakages were found with $1.5 \cdot 10^7$ measurements in this case. By contrast, the second-order leakage detection presented in Figure 11c is successful after 10^6 traces (i.e. differences in the variances are found). We note that the pre-processing required to run second-order detection is greatly simplified thanks to the knowledge of the design (as discussed in [13]). In our case, the



(a) First order, PRG off, $5 \cdot 10^3$ traces.

(b) First order, PRG on, $1.5 \cdot 10^7$ traces.

(c) Second order, PRG on, $1 \cdot 10^6$ traces.

Fig. 11: t-test results. The red horizontal line is the 4.5 threshold. The blue horizontal line is the threshold set based on [24].

[7] The case of an adversary targeting directly the value and not a transition has also been studied. In that case, the information available is much lower. Since we aim at worst-case evaluations, only the results of transition-based attacks are reported.

[8] `http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html`

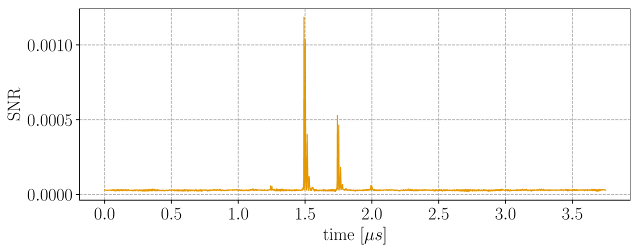
accesses to two shares of $\boldsymbol{msk}$ are always distant by a constant $m = 128$ cycles. This makes the non-linear combination of the shares trivial.

Note also that as in general with leakage detection, these results do not necessarily imply that there are no first-order leakages, but only that the second-order leakages are anyway leading to the most efficient attack [27]. Note that we did not repeat this security order evaluation for the DOM architecture since it was shown to satisfy the independence condition to a sufficient extent for low number of shares in [36]. The investigation of larger number of shares and security orders is an interesting scope for further research (in particular the investigation whether the higher time and space separation of the shares in a LWR-based re-keying may lead to better security guarantees). Finally, the independence assumption for LWR should scale at any order by design since there is no change in the internal design just more repetitions. This is very different with block ciphers where internal cipher computations have to be modified (which may impact independence).

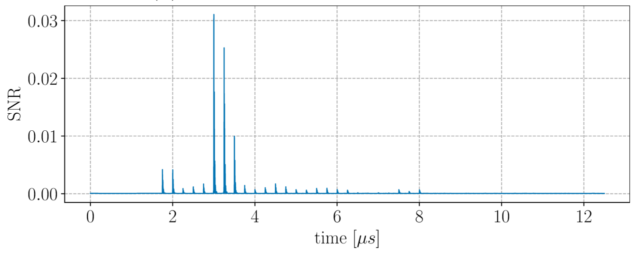
5.3 Bounding mutual information

The second step in our security evaluation is to bound the MI between a single share and the leakage traces. To do so, we compute the HI, which is proven to be an upper bound of the MI in [12], making it suitable for worst-case evaluations. More precisely, we first computed the univariate bounds for individual leakage samples and verified that Gaussian templates lead to similar amounts of extracted information as (non-parametric) histograms. We then considered the multivariate extension for the Gaussian case only, next denoted as gHI, which is computed based on a leakage model $\hat{\mathsf{L}}$ with Equation (8). We performed this analysis for the LWR-based re-keying and the AES-DOM architecture, with a total of 10^7 measurements in both cases.

Leakage model estimation. In order to compute the gHI, the designer has to build a leakage model $\hat{\mathsf{L}}(x, t)$. For this purpose, we first determined the attack surface by estimating a SNR [44]. In short, this simple metric highlights the dependency of the leakage traces to a given value (i.e. a share). It can be efficiently computed for 8-bits values and, under a Gaussian assumption, is directly related to the univariate MI [43]. The (8-bit) SNR of a single share at every time sample is given in Figure 12. The peaks in the plots indicate the time samples depending on the targeted share. The SNR of a share of the LWR-based re-keying (resp. AES-DOM architecture) is depicted in Figure 12a (resp. Figure 12b), leading to two main observations.



(a) LWR: multiplication output.



(b) AES-DOM: Sbox output.

Fig. 12: Signal-to-Noise Ratio (SNR).

First, there are less peaks of SNR exploitable by the adversary in the LWR-based re-keying than in the AES-DOM architecture. Roughly, sensitive information is manipulated during 4 cycles in the first case and during 22 cycles in the second one. As a result, we can expect that multivariate attacks will lead to higher gains in the AES-DOM case.

Second, the best SNR for the LWR-based re-keying is one order of magnitude lower than the one of the AES-DOM. The tentative explanation for this difference is FPGA-specific and relates to the types of FPGA resources that are used in both designs. For the AES-DOM architecture, the Sbox is implemented within registers and LUTs. In the case of the LWR-based re-keying, the multiplication is performed in DSPs which are optimized (hard-coded) cells leading to a smaller power consumption. So the interpretation of this difference is contrasted. On the one hand, it may not be observed for all devices. On the other hand, it is made possible by the simple/standard operations that the LWR-based re-keying exploits (and in particular, the fact that it does not require communications between the shares' computations at the gate level).

Based on these findings, a multivariate leakage model $\hat{\mathsf{L}}(x, t)$ was estimated for a single share. This model corresponds to the mean of the leakage at time t when the share x is manipulated. In the case of an 8-bit adversary (e.g. against the AES-DOM architecture), the 2^8 means corresponding to all the possible x's can directly be estimated. In the case of a 32-bit adversary (e.g. against the LWR-based re-keying), such a direct approach is computationally intensive. As a result, we rather estimated the leakage model thanks to linear regression [53]. Precisely, the leakage is then approximated as

$$\hat{\mathsf{L}}(x,t) = \alpha_0(t) + \sum_{i=1}^{n} \alpha_i(t) \cdot b_i, \tag{9}$$

where b_is denote the i^{th}-bit of x and $\alpha_i(t)$s the bits' weights at time t. We computed such a leakage model for the 4 cycles showing a significant SNR in the LWR-based re-keying traces. The α_i's as a function of the time are given in Appendix A, Figure 16 for completeness.

Security bounds. The upper MI bounds computed with the previously obtained models are shown in Figure 13a for multivariate adversaries. On that plot, the X-axis denotes the number of dimensions n exploited by the adversary. The Y-axis is the gHI. It again leads to two main observations.

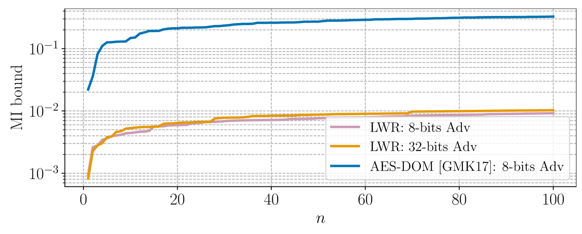
First, already for $n = 1$ the information extracted by the adversary is one order of magnitude larger for the AES-DOM architecture than for LWR-based re-keying, which confirms the intuition of the best SNR in Figure 12.

Second, the figure confirms the significant adversarial gains obtained when moving from univariate attacks to multivariate ones, and that these gains are more important in the AES-DOM case (due to the larger amount of leaking samples), as illustrated in Figure 13b.

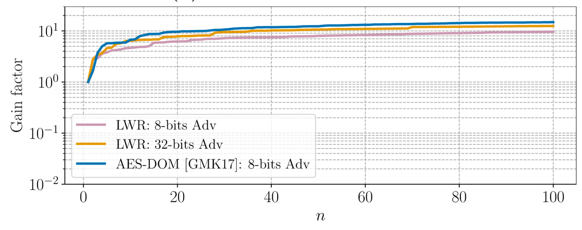
For completeness, we also report the results for an 8-bit adversary against the LWR-based re-keying (which as expected, leads to less information extracted). Overall, these results confirm the good implementation features of the LWR-based re-keying. Namely, its opportunities to exploit optimized (hard-coded) FPGA blocks reduce the univariate leakages, and its minimum manipulation of each share reduces the risks of horizontal attacks.

Data complexity extrapolations. With the previous MI bounds and assuming that the shares' independence is verified at all orders, worst-case attack complexities can be easily lower bounded, as finally given in Figure 14 for univariate and multivariate attacks. The figure reports the evolution of the number of measurements necessary to mount an attack according to the number of shares in the masking schemes and leads to the following observations.

Considering that only univariate attacks are possible (which is undesirable), 6 shares are sufficient for the LWR-based re-keying to require a worst-case attack data complexity larger than 2^{64}. In the case of the AES-DOM, 11 shares are required to reach this security level, which is due to a higher MI per share. Considering multivariate adversaries, similar data complexities are obtained by using 9 and 38 shares for the LWR-based re-keying and the AES-DOM, respectively. The

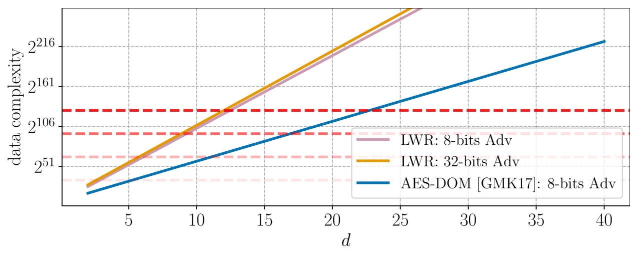


(a) Multivariate bound.

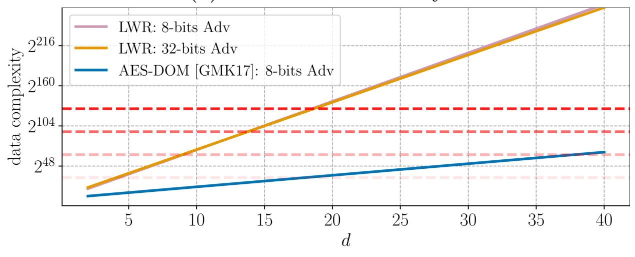


(b) Gain of multivariate adversary over a univariate one.

Fig. 13: Upper bound to MI obtained from 10^7 traces.



(a) Univariate adversary.



(b) Multivariate adversary.

Fig. 14: Lower bound to attack complexity. Dashed lines are multiple de $32 - bit$ security.

additional security gain of the LWR-based re-keying is then due to its more limited vulnerability to horizontal attacks.

6 Conclusions

Putting security and performance evaluations together, Figure 15 highlights the good implementation features of a LWR-based re-keying and the different tradeoffs it provides compared to the DOM masking scheme applied to the AES.

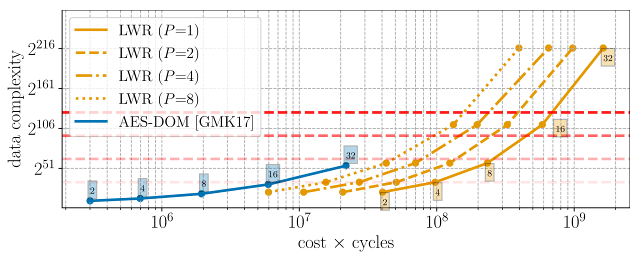
On the one hand and as expected, direct comparisons are not directly favorable and performance gains (in our evaluation setting) can only materialize for high number of shares and assuming high-quality random numbers.[9]

On the other hand, such direct comparisons ignore various other advantages of the LWR-based re-keying. First, the simplicity of the design makes it an appealing choice for developers having limited experience in the implementation of masking schemes, or having limited time for advanced optimizations and evaluations. Second, it carries significantly lower risks of different types: (*i*) compositional flaws are prevented by a simple refreshing strategy, while such flaws have been identified in various masked block cipher implementations [48]; (*ii*) risks of couplings in LWR-based re-keying architectures are strongly mitigated thanks to a strong physical separation between the shares, which is not possible in the case of masked block ciphers where the secure execution of small parts of circuit (e.g. gates) require interaction (so proximity in time and space); (*iii*) multivariate/horizontal attacks are made difficult by the minimum manipulation of each share, while the amount of shares' manipulations increases with d in masked block ciphers (typically requiring larger amounts of noise as d increases, or more expensive gadgets that are secure against horizontal attacks [5,16]).

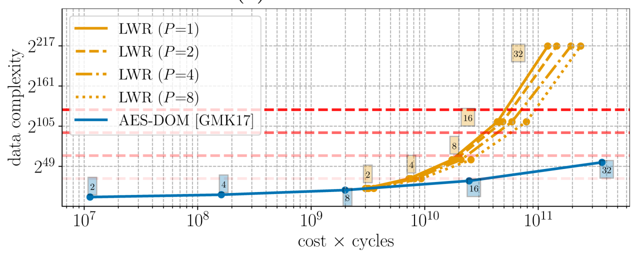
Admittedly, compositional flaws can be prevented by a sound refreshing strategy (which is not significantly more expensive than DOM [15]) and the impact of physical couplings for high security orders is still to be investigated. But those remain potential risks of security reductions. More importantly, the better resistance against horizontal attacks is a significant advantage, since it allows leveraging the same amount of noise for any security level. In this respect, we finally want to emphasize that our lower security bound is pessimistic for the LWR-based re-keying, since it is based on a setup with $P = 1$ (i.e. no algorithmic noise). In this respect, considering larger P's should roughly divide the MI per share by P and therefore increase the attacks' data complexity by a factor P^d. We illustrate this claim with Figure 18 in Appendix C which extrapolates our results by a assuming a linear impact of P on the SNR. As expected, it shows that as the security order increases, such an additive noise is increasingly beneficial (as similar trend holds for any masking scheme).

So overall, despite our evaluations show clear performance overheads for the LWR-based re-keying compared to a standard block cipher protected with DOM, these overheads remain limited, and come with significant advantages in terms of simplicity of design and evaluation, and in terms of risks.

[9] Which (again) calls for a better understanding of the exact requirements for the randomness used in masking schemes.



(a) LFSR PRG.



(b) LR-PRG.

Fig. 15: cost×cycles metric versus data complexity for a multivariate adversary (univariate in Appendix B, Figure 17). Numbers correspond to the number of shares within the implementation.

Acknowledgement: François-Xavier Standaert is a Senior Research Associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the European Union (EU) through the ERC project 724725 (acronym SWORD).

References

1. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.: On the cost of lazy engineering for masked software implementations. In: CARDIS, *LNCS*, vol. 8968, pp. 64–81. Springer (2014)
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified proofs of higher-order masking. In: EUROCRYPT (1), *LNCS*, vol. 9056, pp. 457–485. Springer (2015)
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: ACM Conference on Computer and Communications Security, pp. 116–129. ACM (2016)
4. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel implementations of masking schemes and the bounded moment leakage model. In: EUROCRYPT (1), *LNCS*, vol. 10210, pp. 535–566 (2017)
5. Battistello, A., Coron, J., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: CHES, *LNCS*, vol. 9813, pp. 23–39. Springer (2016)
6. Becker, G., Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A., Marson,

M., Rohatgi, P., et al.: Test vector leakage assessment (TVLA) methodology in practice

7. Belaïd, S., Coron, J., Fouque, P., Gérard, B., Kammerer, J., Prouff, E.: Improved side-channel analysis of finite-field multiplication. In: CHES, *LNCS*, vol. 9293, pp. 395–415. Springer (2015)
8. Belaïd, S., Fouque, P., Gérard, B.: Side-channel analysis of multiplications in $GF(2^{128})$ - application to AES-GCM. In: ASIACRYPT (2), *LNCS*, vol. 8874, pp. 306–325. Springer (2014)
9. Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal verification of masked hardware implementations in the presence of glitches. In: EUROCRYPT (2), *LNCS*, vol. 10821, pp. 321–353. Springer (2018)
10. Bogdanov, A., Rosen, A.: Pseudorandom functions: Three decades later. In: Tutorials on the Foundations of Cryptography, pp. 79–158. Springer International Publishing (2017)
11. Brenner, H., Gaspar, L., Leurent, G., Rosen, A., Standaert, F.: FPGA implementations of SPRING - and their countermeasures against side-channel attacks. In: CHES, *Lecture Notes in Computer Science*, vol. 8731, pp. 414–432. Springer (2014)
12. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.: Leakage certification revisited: Bounding model errors in side-channel security evaluations. In: CRYPTO (1), *Lecture Notes in Computer Science*, vol. 11692, pp. 713–737. Springer (2019)
13. Bronchain, O., Schneider, T., Standaert, F.: Multi-tuple leakage detection and the dependent signal issue. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 318–345 (2019)
14. Burman, S., Mukhopadhyay, D., Veezhinathan, K.: LFSR based stream ciphers are vulnerable to power attacks. In: INDOCRYPT, *Lecture Notes in Computer Science*, vol. 4859, pp. 384–392. Springer (2007)
15. Cassiers, G., Grégoire, B., Levi, I., Standaert, F.: Hardware private circuits: From trivial composition to full verification. IACR Cryptol. ePrint Arch. **2020**, 185 (2020)
16. Cassiers, G., Standaert, F.: 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). DOI 10.13154/tches.v2019.i2.162-198. URL `https://doi.org/10.13154/tches.v2019.i2.162-198`
17. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: CRYPTO, *LNCS*, vol. 1666, pp. 398–412. Springer (1999)
18. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: CHES, *LNCS*, vol. 2523, pp. 13–28. Springer (2002)
19. de Chérisey, E., Guilley, S., Rioul, O., Piantanida, P.: 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). DOI 10.13154/tches.v2019.i2.49-79. URL `https://doi.org/10.13154/tches.v2019.i2.49-79`
20. Cnudde, T.D., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? In: COSADE, *LNCS*, vol. 10348, pp. 1–18. Springer (2017)
21. Cnudde, T.D., Ender, M., Moradi, A.: Hardware masking, revisited. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 123–148 (2018). DOI 10.13154/tches.v2018.i2.123-148. URL `https://doi.org/10.13154/tches.v2018.i2.123-148`
22. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: CHES, *LNCS*, vol. 9813, pp. 194–212. Springer (2016)
23. Coron, J., Giraud, C., Prouff, E., Renner, S., Rivain, M., Vadnala, P.K.: Conversion of security proofs from one leakage model to another: A new issue. In: COSADE, *LNCS*, vol. 7275, pp. 69–81. Springer (2012)
24. Ding, A.A., Zhang, L., Durvaux, F., Standaert, F., Fei, Y.: Towards sound and optimal leakage detection procedure. In: CARDIS, *LNCS*, vol. 10728, pp. 105–122. Springer (2017)
25. Dobraunig, C., Koeune, F., Mangard, S., Mendel, F., Standaert, F.: Towards fresh and hybrid re-keying schemes with beyond birthday security. In: CARDIS, *LNCS*, vol. 9514, pp. 225–241. Springer (2015)
26. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: EUROCRYPT, *LNCS*, vol. 8441, pp. 423–440. Springer (2014)
27. Duc, A., Faust, S., Standaert, F.: Making masking security proofs concrete - or how to evaluate the security of any leaking device. In: EUROCRYPT (1), *LNCS*, vol. 9056, pp. 401–429. Springer (2015)
28. Durvaux, F., Standaert, F.: From improved leakage detection to the detection of points of interests in leakage traces. In: EUROCRYPT (1), *LNCS*, vol. 9665, pp. 240–262. Springer (2016)
29. Dziembowski, S., Faust, S., Herold, G., Journault, A., Masny, D., Standaert, F.: Towards sound fresh re-keying with hard (physical) learning problems. In: CRYPTO (2), *LNCS*, vol. 9815, pp. 272–301. Springer (2016)
30. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 89–120 (2018). DOI 10.13154/tches.v2018.i3.89-120. URL `https://doi.org/10.13154/tches.v2018.i3.89-120`
31. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: CHES, *LNCS*, vol. 2162, pp. 251–261. Springer (2001)
32. Gaspar, L., Leurent, G., Standaert, F.: Hardware implementation and side-channel analysis of lapin. In: CT-RSA, *Lecture Notes in Computer Science*, vol. 8366, pp. 206–226. Springer (2014)
33. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation
34. Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? In: EUROCRYPT (1), *LNCS*, vol. 10210, pp. 567–597 (2017)
35. Groß, H., Mangard, S.: A unified masking approach. J. Cryptographic Engineering **8**(2), 109–124 (2018). DOI 10.1007/s13389-018-0184-y. URL `https://doi.org/10.1007/s13389-018-0184-y`
36. Groß, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: CT-RSA, *LNCS*, vol. 10159, pp. 95–112. Springer (2017)
37. Grosso, V., Standaert, F.: Masking proofs are tight and how to exploit it in security evaluations. In: EUROCRYPT (2), *LNCS*, vol. 10821, pp. 385–412. Springer (2018)

38. Guo, Q., Johansson, T.: A new birthday-type algorithm for attacking the fresh re-keying countermeasure. Inf. Process. Lett. **146**, 30–34 (2019). DOI 10.1016/j.ipl.2019.02.005. URL `https://doi.org/10.1016/j.ipl.2019.02.005`
39. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: CRYPTO, *LNCS*, vol. 2729, pp. 463–481. Springer (2003)
40. Journault, A., Standaert, F.: Very high order masking: Efficient implementation and security evaluation. In: CHES, *LNCS*, vol. 10529, pp. 623–643. Springer (2017)
41. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO, *LNCS*, vol. 1666, pp. 388–397. Springer (1999)
42. Levi, I., Bellizia, D., Standaert, F.: Reducing a masked implementation's effective security order with setup manipulations and an explanation based on externally-amplified couplings. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 293–317 (2019)
43. Mangard, S., Oswald, E., Standaert, F.: One for all - all for one: unifying standard differential power analysis attacks. IET Information Security **5**(2), 100–110 (2011). DOI 10.1049/iet-ifs.2010.0096. URL `https://doi.org/10.1049/iet-ifs.2010.0096`
44. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: CT-RSA, *LNCS*, vol. 3376, pp. 351–365. Springer (2005)
45. Mangard, S., Pramstaller, N., Oswald, E.: Successfully attacking masked AES hardware implementations. In: CHES, *LNCS*, vol. 3659, pp. 157–171. Springer (2005)
46. Mather, L., Oswald, E., Whitnall, C.: Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In: ASIACRYPT (1), *LNCS*, vol. 8873, pp. 243–261. Springer (2014)
47. Medwed, M., Standaert, F., Großschädl, J., Regazzoni, F.: Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In: AFRICACRYPT, *LNCS*, vol. 6055, pp. 279–296. Springer (2010)
48. Moos, T., Moradi, A., Schneider, T., Standaert, F.: Glitch-resistant masking revisited or why proofs in the robust probing model are needed. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 256–292 (2019). DOI 10.13154/tches.v2019.i2.256-292. URL `https://doi.org/10.13154/tches.v2019.i2.256-292`
49. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In: EUROCRYPT, *LNCS*, vol. 6632, pp. 69–88. Springer (2011)
50. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptology **24**(2), 292–321 (2011). DOI 10.1007/s00145-010-9085-7. URL `https://doi.org/10.1007/s00145-010-9085-7`
51. Pereira, O., Standaert, F., Vivek, S.: Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In: ACM Conference on Computer and Communications Security, pp. 96–108. ACM (2015)
52. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: EUROCRYPT, *LNCS*, vol. 7881, pp. 142–159. Springer (2013)
53. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: CHES, *LNCS*, vol. 3659, pp. 30–46. Springer (2005)
54. Schneider, T., Moradi, A.: Leakage assessment methodology - extended version. J. Cryptographic Engineering **6**(2), 85–99 (2016). DOI 10.1007/s13389-016-0120-y. URL `https://doi.org/10.1007/s13389-016-0120-y`
55. Standaert, F., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: EUROCRYPT, *Lecture Notes in Computer Science*, vol. 5479, pp. 443–461. Springer (2009)
56. Standaert, F.X.: How (not) to use welch's t-test in side-channel security evaluations. In: CARDIS, pp. 65–79. Springer (2018)
57. Welch, B.L.: The generalization ofstudent's' problem when several different population variances are involved. Biometrika **34**(1/2), 28–35 (1947)
58. Yiu, J.: The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors. Newnes (2013)

A Leakage model

Linear basis used as a leakage model. Each of the curves corresponds to a single α_i showing the activity of each of the bits across time.

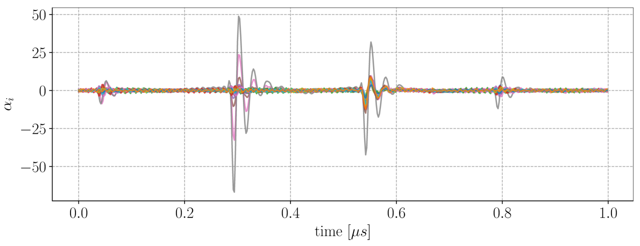
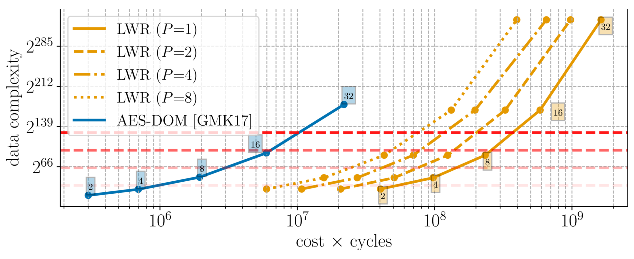


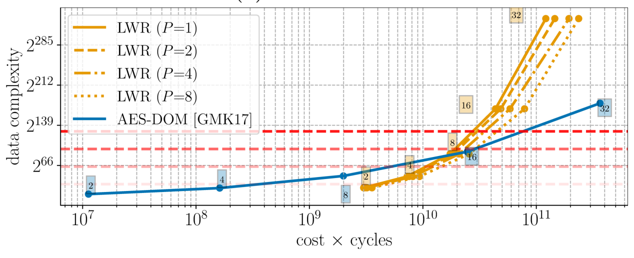
Fig. 16: Linear basis for LWR leakage model.

B Cost of security

C Parallelism parameter

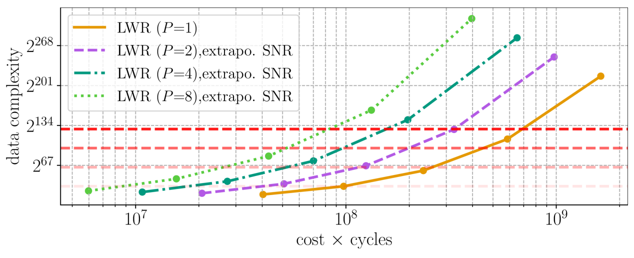


(a) LFSR PRG.

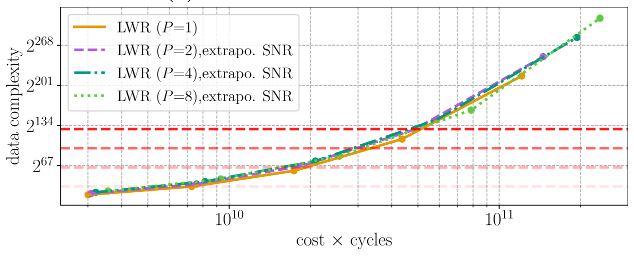


(b) Leakage resilient PRG.

Fig. 17: cost×cycles metric versus data complexity for a univariate adversary. Numbers correspond to the number of shares within the implementation.



(a) LFSR PRG. Multivariate



(b) Leakage resilient PRG. Multivariate

Fig. 18: cost×cycles metric versus data complexity for a univariate and multivariate adversaries with a linear extrapolation on the noise with P.