Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model

Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, Pierre-Yves Strub

Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model

Gilles Barthe[1], François Dupressoir[2], Sebastian Faust[3], Benjamin Grégoire[4], François-Xavier Standaert[5], and Pierre-Yves Strub[6].

[1] IMDEA Software Institute, Spain.
[2] University of Surrey, UK.
[3] Ruhr Universität Bochum, Germany.
[4] Inria Sophia-Antipolis – Méditerranée, France.
[5] Université Catholique de Louvain, Belgium.
[6] Ecole Polytechnique, France.

Abstract. In this paper, we provide a necessary clarification of the good security properties that can be obtained from parallel implementations of masking schemes. For this purpose, we first argue that (i) the probing model is not straightforward to interpret, since it more naturally captures the intuitions of serial implementations, and (ii) the noisy leakage model is not always convenient, e.g. when combined with formal methods for the verification of cryptographic implementations. Therefore we introduce a new model, the bounded moment model, that formalizes a weaker notion of security order frequently used in the side-channel literature. Interestingly, we prove that probing security for a serial implementation implies bounded moment security for its parallel counterpart. This result therefore enables an accurate understanding of the links between formal security analyses of masking schemes and experimental security evaluations based on the estimation of statistical moments Besides its consolidating nature, our work also brings useful technical contributions. First, we describe and analyze refreshing and multiplication algorithms that are well suited for parallel implementations and improve security against multivariate side-channel attacks. Second, we show that simple refreshing algorithms (with linear complexity) that are not secure in the continuous probing model are secure in the continuous bounded moment model. Eventually, we discuss the independent leakage assumption required for masking to deliver its security promises, and its specificities related to the serial or parallel nature of an implementation.

1 Introduction

The masking countermeasure is currently the most investigated solution to improve security against power-analysis attacks [**?**]. It has been analyzed theoretically in the so-called probing and noisy leakage models [**?**,**?**], and based on a large number of case studies, with various statistical tools (e.g. [**?**,**?**] for non-profiled and profiled attacks, respectively). Very briefly summarized, state-of-the-art masking schemes are currently divided in two main trends: on the one hand, software-oriented masking, following the initial work of Prouff and Rivain [**?**]; on the other hand hardware-oriented masking (or *threshold implementations*) following the inital work of Nikova, Rijmen and Schläffer [**?**].

At CRYPTO 2015, Reparaz et al. highlighted interesting connections between the circuit constructions in these two lines of works [**?**]. Looking at these links, a concrete difference remains between software- and hardware-oriented masking schemes. Namely, the (analyses of the) first ones usually assume a serial manipulation of the shares while the (implementations of the) second ones encourage their parallel manipulation.[1] Unfortunately, the probing leakage model, that has led to an accurate understanding of the security guarantees of software-oriented masking schemes [**?**], is not directly interpretable in the parallel setting. Intuitively, this is because the parallel manipulation of the shares reveals information on all of them, e.g. via their sum, but observing sums of wires is not permitted in the probing model. As will be clear in the following, this does not limit the concrete relevance of the probing model. Yet, it reveals a gap between the level of theoretical understanding of serial and parallel masked implementations.

1.1 Our contribution

Starting from the observation that parallelism is a key difference between software and hardware-oriented masking, we introduce a new model – the bounded moment model – that allows rigorous reasoning and efficient analyses of parallel masked implementations. In summary, the bounded moment model can be seen as the formal counterpart to the notion of security against higher-order attacks [**?**,**?**], just as the noisy leakage model [**?**] is the formal counterpart to information theoretic leakage metrics such as introduced in [**?**]. It allows us to extend the consolidating work of [**?**] and to obtain the following results:

- First, we exhibit a natural connection between the probing model and the bounded moment model. More precisely, we prove that security in the probing model for a serial implementation implies security in the bounded moment model for the corresponding parallel implementation.
- Next, we propose regular refreshing and multiplication algorithms suitable for parallel implementations. Thanks to parallelism, these algorithms can be implemented in linear time, with the same memory requirements as a serial implementation (since masking requires to store all the shares anyway). Note that the refreshing algorithm is particularly appealing for combination with key-homomorphic primitives (e.g. inner product based [**?**]), since it allows them to be masked with linear (time and randomness) complexity. As for the multiplication algorithm, its linear execution time also provides improved security against multivariate (aka horizontal) side-channel attacks [**?**].
- Third, we exhibit the concrete separation between the probing model and the bounded moment model. For this purpose, we provide simple examples from the literature on leakage squeezing and low-entropy masking schemes showing that (for linear leakage funtions) it is possible to have a larger security

[1] This division between hardware and software is admittedly oversimplifying in view of the improved capabilities of modern microprocessors to take advantage of parallelism. So the following results in fact also apply to parallel software computing.

order in the bounded moment model than in the probing model [?,?]. More importantly, we show that our simple refreshing algorithm is insecure in the probing model against adversaries taking advantage of continuous leakage, while it remains secure against such (practically relevant) adversaries in the bounded moment model. This brings a theoretical foundation to the useful observation that simple refreshing schemes that are sometimes considered in practice (e.g. adding shares that sum to zero) do not lead to devastating attacks when used to refresh an immutable secret state (e.g. a block cipher key), despite their lack of security in the continuous probing model. Note that the latter result is also of interest for serial implementations.
- Finally, we illustrate our results with selected case studies, and take advantage of them to discuss the assumption of independent leakages in side-channel attacks (together with its underlying physical intuitions).

1.2 Related work

Serial masking and formal methods. The conceptual simplicity of the probing model makes it an attractive target for automated verification. Recognizing the close similarities between information-flow policies and security in the probing model, Moss, Oswald, Page and Turnstall [?] build a masking compiler that takes as input an unprotected program and outputs an equivalent program that resists first-order DPA. Their compiler performs a type-based analysis of the input program and iteratively transforms the program when encountering a typing error. Aiming for increased generality, Bayrak, Regazzoni, Novo and Ienne [?] propose a SMT-based method for analyzing statistical independence between secret inputs and intermediate computations, still in the context of first-order DPA. In a series of papers starting with [?], Eldib, Wang and Schaumont develop more powerful SMT-based methods for synthesizing masked implementations or analyzing the security of existing masked implementations. Their approach is based on a logical characterization of security at arbitrary orders in the probing model. In order to avoid the "state explosion" problem, which results from looking at higher-orders and from the logical encoding of security in the probing model, they exploit elaborate methods that support incremental verification, even for relatively small orders. A follow-up by Eldib and Wang [?] extends this idea to synthesize masked implementations fully automatically. Leveraging the connection between probabilistic information flow policies and relational program logics, Barthe, Belaïd, Dupressoir, Fouque, Grégoire and Strub [?] introduce another approach based on a domain-specific logic for proving security in the probing model. Like Eldib, Wang and Schaumont, their method applies to higher orders. Interestingly, it achieves practicality at orders up to four for multiplications and S-boxes. In a complementary line of work, Belaïd, Benhamouda, Passelegue, Prouff, Thillard and Vergnaud [?] develop an automated tool for finding probing attacks on implementations and use it to discover optimal (in randomness complexity) implementations of multiplication at order 2, 3, and 4 (with 2, 4, and 5 random bits). They also propose a multiplication for arbitrary orders, requiring $\frac{d^2}{4} + d$ bits of randomness to achieve security at order d.

All these works focus on the usual definition of security in the probing model. In contrast, Barthe, Belaïd, Dupressoir, Fouque and Grégoire introduce a stronger notion of security, called strong non-interference (or SNI), which enables compositional verification of higher-order masking schemes [**?**], and leads to much improved capabilities to analyze large circuits (i.e. full algorithms, typically). Similar to several other security notions for the probing model, strong non-interference is qualitative, in the sense that a program is either secure or insecure. Leaving the realm of qualitative notions, Eldib, Wang, Taha, and Schaumont [**?**] consider a quantitative relaxation of the usual definition of (probing) security, and adapt their tools to measure the quantitative masking strength of an implementation. Their definition is specialized to first-order moments, but the connections with the bounded moment model are evident, and it would be interesting to explore generalizations of their work to our new model.

Threshold and parallel implementations. The inital motivation of Nikova, Rijmen and Schläffer was the observation that secure implementations of masking in hardware are challenging, due to the risk of glitches recombining the shares [**?**]. Their main idea to prevent this issue is to add a condition of non-completeness to the masked computations (i.e. ensure that any combinatorial circuit never takes all shares as input). Many different works have confirmed the practical relevance of this additional requirement, making it the de facto standard for hardware masking (see [**?**,**?**,**?**,**?**,**?**] for a few examples). Our following results are particularly relevant to threshold implementations since (*i*) in view of their hardware specialization, they encourage a parallel manipulation of the shares, (*ii*) most of their security evaluations so far were based on the estimation of statistical moments that we formalize with the bounded moment model, and (*iii*) their higher-order implementations suggested in [**?**] and recently analyzed in [**?**] exploit the simple refreshing scheme that we study in Section 8.2.

Noisy leakage model. Note that the noisy leakage model in [**?**] also provides a natural way to capture parallel implementations (and in fact a more general one: see Figure 7 in conclusions). Yet, this model is not always convenient when exploiting the aforementioned formal methods. Indeed, these tools benefit greatly from the simplicity of the probing model in order to analyze complex implementations, and hardly allow the manipulation of noisy leakages. In this respect, the bounded moment model can be seen as a useful intermediate (i.e. bounded moment security can be efficiently verified with formal methods, although its verification naturally remains slower than probing security).

Eventually, we believe it is fundamentally interesting to clarify the connections between the mainstream (probing and) noisy leakage model(s) and concrete evaluation strategies based on the estimation of statistical moments. In this respect, it is the fact that bounded moment security requires a weaker independence condition than probing security that enables us to prove the simple refreshing of Section 8.2, which is particularly useful in practice, especially compared to previous solutions for efficient refreshing algorithms such as [**?**]. Here as well, directly dealing with noisy leakages would be more complex.

2 Background

In this section, we introduce our leakage setting for serial and parallel implementations. Note that for readability, we keep the description of our serial and parallel computing models informal, and defer their definition to Section 5.

2.1 Serial implementations

We start from the description of leakage traces in [**?**], where y is a n-bit sensitive value manipulated by a leaking device. Typically, it could be the output of an S-box computation such that $y = \mathsf{S}(x \oplus k)$ with n-bit plaintext and key words x and k. Let $y_1, y_2, \ldots, y_d$ be the d shares representing y in a Boolean masking scheme (i.e. $y = y_1 \oplus y_2 \oplus \ldots \oplus y_d$). In a side-channel attack, the adversary is provided with some information (or leakage) on each share. Concretely, the type of information provided highly depends on the type of implementation considered. For example, in a serial implementation, we typically have that each share is manipulated during a different "cycle" c so that the number of cycles in the implementation equals the number of shares, as in Figure 1(a). The leakage in each cycle then takes the form of a random variable $\boldsymbol{L}_c$ that is the output of a leakage function L_c, which takes y_c and a noise variable $\boldsymbol{R}_c$ as arguments:

$$\boldsymbol{L}_c = \mathsf{L}_c(y_c, \boldsymbol{R}_c) \ , \quad \text{with } 1 \leq c \leq d \ . \tag{1}$$

That is, each subtrace $\boldsymbol{L}_c$ is a vector, the elements of which represent time samples. When accessing a single sample τ, we use the notation $L_c^\tau = \mathsf{L}_c^\tau(y_c, \boldsymbol{R}_c)$. From this general setup, a number of assumptions are frequently used in the literature on side-channel cryptanalysis. We consider the following two (also considered in [**?**]). First, we assume that the leakage vectors $\boldsymbol{L}_c$ are independent random variables. This a strict requirement for masking proofs to hold and will be specifically discussed in Section 9. Second, and for convenience only, we assume that the leakage functions are made of a deterministic part $\mathsf{G}_c(y_c)$ and additive noise $\boldsymbol{R}_c$ so that $\boldsymbol{L}_c = \mathsf{L}_c(y_c, \boldsymbol{R}_c) \approx \mathsf{G}_c(y_c) + \boldsymbol{R}_c$. Note that the $+$ symbol here denotes the addition in $\mathbb{R}$ (while $\oplus$ denotes a bitwise XOR).

2.2 Parallel implementations

We now generalize the previous serial implementation to the parallel setting. In this case, the main difference is that several shares can be manipulated in the same cycle. For example, the right part of Figure 1 shows the leakage corresponding to a fully parallel implementation where all the shares are manipulated in a single cycle. As a result, we have a single leakage vector $\boldsymbol{L}_1 = \mathsf{L}(y_1, y_2 \ldots, y_d, \boldsymbol{R}_1)$. More generally, we will consider N-cycle parallel implementations such that for each cycle c ($1 \leq c \leq N$), we define the set of shares that are manipulated during the cycle as $\mathcal{Y}_c$, and the number of shares in a set $\mathcal{Y}_c$ as n_c. This means that a masked implementation requires at least that the union of these sets equals $\{y_1, y_2, \ldots, y_d\}$, i.e. all the shares need to be manipulated at least once. This

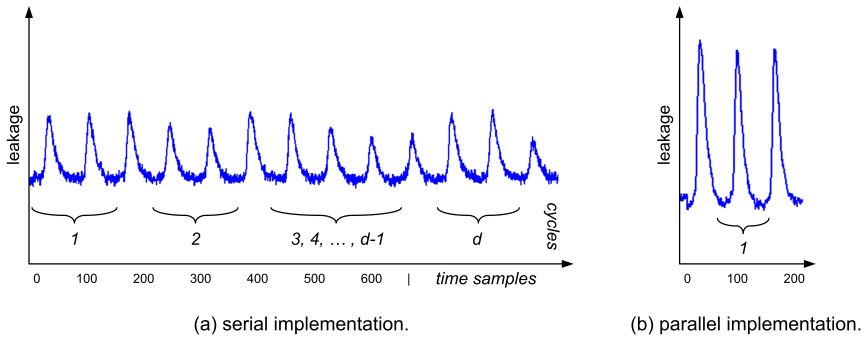


Fig. 1. Leakage trace of d-shared secret.

model of computation is a generalization of the previous one since the serial implementation in the left part of the figure is simply captured with the case $N = d$ and, for every c, $n_c = 1$. As previously mentioned, the highly parallel implementation in the right part of the figure is captured with the case $N = 1$ and $n_1 = d$. For simplicity, we refer to this case as the parallel implementation case in the following. Any intermediate solution mixing serial and parallel computing (e.g. 2 shares per cycle, 3 shares per cycle, ...) can be captured by our model. Concretely, the impact of parallel computation is reflected both by a reduced number of cycles and by an increased instantaneous power consumption, illustrated with the higher amplitude of the curves in Figure 1(b). A simple abstraction to reflect this larger power consumption is the following linear model:

$$\boldsymbol{L}_c = \alpha_c^1 \cdot \mathsf{G}_c^1\left(\mathcal{Y}_c(1)\right) + \alpha_c^2 \cdot \mathsf{G}_c^2\left(\mathcal{Y}_c(2)\right) + \ldots + \alpha_c^{n_c} \cdot \mathsf{G}_c^{nc}\left(\mathcal{Y}_c(n_c)\right) + \boldsymbol{R}_c. \quad (2)$$

with all α_c^j's $\in \mathbb{R}$. Contrary to the additive noise assumption that is only used for convenience and not needed for masking proofs, this linear model is a critical ingredient of our analysis of parallel implementations, since it is needed to maintain the independent leakage assumption. As for other physical issues that could break this assumption, we assume Equation (2) holds in the next sections and discuss its possible limitations in Section 9. Yet, we already note that a general contradiction of this hypothesis would imply that any (e.g. threshold) implementation manipulating its shares in parallel should be insecure.

3 Security models

3.1 Probing security and noisy leakage

We first recall two important models for analyzing masking countermeasures.

First, the conceptually simple *t-probing* and *ϵ-probing* (or random probing) models were introduced in [**?**]. In the former, the adversary obtains t intermediate values of the computation (e.g. can probe t wires if we compute in binary fields). In the latter, he rather obtains each of these intermediate values with probability

ϵ, and gets $\perp$ with probability $1-\epsilon$ (where $\perp$ means no knowledge). Using a Chernoff-bound, it is easy to show that security in the t-probing model reduces to security in the ϵ-probing model for certain values of ϵ.

Second, the noisy leakage model describes many realistic side-channel attacks where an adversary obtains each intermediate value perturbed with a "δ-noisy" leakage function [**?**]. A leakage function L is called δ-noisy if for a uniformly random variable Y we have $\mathrm{SD}(Y; Y|\boldsymbol{L}_Y) \leq \delta$, with SD the statistical distance. It was shown in [**?**] that an equivalent condition is that the leakage is not too informative, where informativity is measured with the standard notion of mutual information $\mathrm{MI}(Y; \boldsymbol{L}_Y)$. In contrast with the ϵ-probing model, the adversary obtains noisy leakage for each intermediate variable. For example, in the context of masking, he obtains $\mathsf{L}(Y_i, \boldsymbol{R}_i)$ for all the shares Y_i, which is reflective of actual implementations where the adversary can potentially observe the leakage of all these shares, since they are all present in leakage traces (as in Figure 1).

Recently, Duc et al. showed that security against probing attacks implies security against noisy leakages [**?**]. This result leads to the natural strategy of proving security in the (simpler) probing model while stating security levels based on the concrete information leakage evaluations (as discussed in [**?**]).

3.2 The bounded moment model

Motivation. In practice, the probing model is perfectly suited to proving the security of the serial implementations from Section 2.1. This is because it ensures that an adversary needs to observe d shares with his probes to recover secret information. Since in a serial implementation, every share is manipulated in a different clock cycle, it leads to a simple analogy between the number of probes and the number of cycles exploited in the leakage traces. By contrast, this simple analogy no longer holds for parallel implementations, where all the shares manipulated during a given cycle can leak concurrently. Typically, assuming that an adversary can only observe a single share with each probe is counter-intuitive in this case. For example, it would be natural to allow that he can observe the output of Equation (2) with one probe, which corresponds to a single cycle in Figure 1(b) and already contains information about all the shares (if $n_c = d$).

As mentioned in introduction, the noisy leakage model provides a natural solution to deal with the leakages of parallel implementations. Indeed, nothing prevents the output of Equation (2) from leaking only a limited amount of information if a large enough noise is considered. Yet, directly dealing with noisy leakages is sometimes inconvenient for the analysis of masked implementations, e.g. when it comes to verification with the formal methods listed in Section 1.2. In view of their increasing popularity in embedded security evaluation, this creates a strong incentive to come up with an alternative model allowing both the construction of proofs for parallel implementations and their efficient evaluation with formal methods. Interestingly, we will show in Section 5 that security in this alternative model is implied by probing security. It confirms the relevance of the aforementioned strategy of first proving security in the probing model, and then stating security levels based on concrete information leakage evaluations.

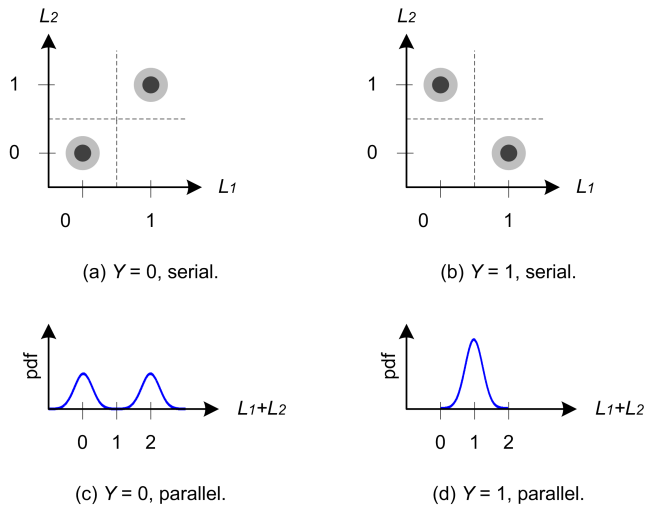


Fig. 2. Leakage distributions of a single-bit 2-shared secret.

Definition. Intuitively, the main limitation of the noisy leakage model in the context of formal methods is that it involves the (expensive) manipulation of complete leakage distributions. In this respect, one natural simplification that fits to a definition of "order" used in the practical side-channel literature is to relate security to the smallest key-dependent statistical moment in the leakage distributions. Concretely, the rationale behind this definition is that the security of a masked implementation comes from the need to estimate higher-order statistical moments, a task that becomes exponentially difficult in the number of shares if their leakages are independent and sufficiently noisy (see the discussion in [**?**]). Interestingly, such a definition directly captures the parallel implementation setting, as can easily be illustrated with an example. Say we have a single-bit sensitive value Y that is split in $d = 2$ shares, and that an adversary is able to observe a leakage function where the deterministic part is the Hamming weight function and the noise is normally distributed. Then, the (bivariate) leakage distribution for a serial implementation, where the adversary can observe the leakage of the two shares separately, is shown in the upper part of Figure 2. And the (univariate) leakage distribution for a parallel implementation, where the adversary can only observe the sum of the leakages of the two shares, is shown in the lower part of the figure. In both cases, the first-order moment (i.e. the mean) of the leakage distributions is independent of Y.

In order to define our security model, we therefore need the following definition.

Definition 1 (Mixed moment at orders $o_1, o_2, \ldots, o_r$). *Let $\{X_i\}_{i=1}^r$ be a set of r random variables. The* mixed moment at orders $o_1, o_2, \ldots, o_r$ of $\{X_i\}_{i=1}^r$ *is:*

$$\mathsf{E}(X_1^{o_1} \times X_2^{o_2} \times \ldots \times X_r^{o_r}),$$

where E *denotes the expectation operator and* $\times$ *denotes the multiplication in* $\mathbb{R}$*. For simplicity, we denote the integer* $o = \sum_i o_i$ *as* the order *of this mixed moment. We further say that a mixed moment at order* o *is* m-variate *(or has dimension* m*) if there are exactly* m *non-zero coefficients* o_i*.*

This directly leads to our defintion of security in the bounded moment model.

Definition 2 (Security in the bounded moment model). *Let* $\{\boldsymbol{L}_c\}_{c=1}^N$ *be the leakage vectors corresponding to an* N*-cycle cryptographic implementation manipulating a secret variable* Y*. This implementation is* secure at order o *if all the mixed moments of order up to* o *of* $\{\boldsymbol{L}_c\}_{c=1}^N$ *are independent of* Y*.*[2]

Say for example that we have a sensitive value Y that is split in $d = 3$ shares, for which we leak the same noisy Hamming weights as in Figure 2. In the case of a (fully) parallel implementation, we have only one leakage sample L_1 and security at order 2 requires that $\mathsf{E}(L_1)$ and $\mathsf{E}(L_1^2)$ are independent of Y. In the case of a serial implementation, we have three samples L_1, L_2, L_3 and must show that $\mathsf{E}(L_1)$, $\mathsf{E}(L_2)$, $\mathsf{E}(L_3)$, $\mathsf{E}(L_1^2)$, $\mathsf{E}(L_2^2)$, $\mathsf{E}(L_3^2)$, $\mathsf{E}(L_1 \times L_2)$, $\mathsf{E}(L_1 \times L_3)$ and $\mathsf{E}(L_2 \times L_3)$ are independent of Y. Note that the only difference between this example and concrete implementations is that in the latter case, each cycle would correspond to a leakage vector $\boldsymbol{L}_c$ rather than a single (univariate) sample L_c.

Note also that this definition allows us to clarify a long standing discussion within the cryptographic hardware community about the right definition of security order. That is, the first definitions for secure masking (namely "perfect masking at order o" in [**?**] and "masking at order o" in [**?**]) where specialized to serial implementations, and required that any tuple of o intermediate variables is independent of any sensitive variable in an implementation. For clarity, we will now call this (strong) independence condition "security at order o in the probing model". However, due to its specialization to serial implementation, this definition also leaves a confusion about whether its generalization to parallel implementations should relate to the smallest dimensionality of a key-dependent leakage distribution (i.e. m in our definition) or the smallest order of a key-dependent moment in these distributions (i.e. o in our definition). Concretely, $m \geq o$ in the case of a serial implementation, but only the second solution generalizes to parallel implementations, since for such implementations the dimensionality can be as low as 1 independent of the number of shares. Hence, we adopt this solution in the rest of the paper and will call this (weaker) independence condition "security at order o in the bounded moment model".

[2] This definition justifies why we use raw moments rather than central or standardized ones. Indeed, to establish security at order o, we require moments of orders less than o to be independent of Y. Thus centralization (i.e. removing the mean) or normalization by the standard deviation only add terms known to be independent of Y.

4 Additional features and discussions

4.1 Experimental model validation

Quite naturally, the introduction of a new leakage model should come with empirical validation that it reasonably matches the peculiarities of actual implementations and their evaluation. Conveniently, in the case of the bounded moment model, we do nothing else than formalizing evaluation approaches that are already deployed in the literature. This is witnessed by attacks based on the estimation of statistical moments, e.g. exploiting the popular difference-of-means and correlation distinguishers [**?**,**?**,**?**]. Such tools have been applied to various protected implementations, including threshold ones [**?**,**?**,**?**,**?**] and other masking schemes or designs running in recent high-frequency devices [**?**,**?**,**?**]. In all these cases, security at order o was claimed if the lowest key-dependent statistical moment of the leakage distribution was found to be of order $o + 1$.

4.2 Dimensionality reduction

One important property of Definition 2 is that it captures security based on the statistical order of the key-dependent moments of a leakage distribution. This means that the dimensionality of the leakage vectors does not affect the security order in the bounded moment model. Therefore, it also implies that such a security definition is not affected by linear dimensionality reductions. This simple observation is formalized by the following definition and lemma.

Definition 3 (Linear dimensionality reduction). *Let $\boldsymbol{L} = [L_1, L_2, \ldots, L_M]$ denote an M-sample leakage vector and $\{\boldsymbol{\alpha}_i\}_{i=1}^{m}$ denote M-element vectors in $\mathbb{R}$. We say that $\boldsymbol{L'} = [L'_1, L'_2, \ldots, L'_m]$ is a linearly reduced leakage vector if each of its (projected) samples L'_i is corresponds to a scalar product $\langle \boldsymbol{L}; \boldsymbol{\alpha}_i \rangle$.*

Lemma 1. *Let $\{\boldsymbol{L}_c\}_{c=1}^{N}$ be the leakage vectors corresponding to an N-cycle cryptographic implementation manipulating a secret variable Y. If this implementation is secure at order o in the bounded moment model, then any implementation with linearly reduced leakages of $\{\boldsymbol{L}_c\}_{c=1}^{N}$ is secure at order o.*

Proof. Since the samples of $\boldsymbol{L}'$ are linear combinations of the samples of $\boldsymbol{L}$, we need the expectation of any polynomial of degree up to o of the samples of $\boldsymbol{L}'$ to be independent of Y. This directly derives from Definition 2 which guarantees that the expectation of any monomial of degree up to o is independent of Y. □

Typical examples of linear dimensionality reductions are PCA [**?**] and LDA [**?**]. Note that while linearly combining leakage samples does not affect bounded moment security, it can be used to reduce the noise of the samples implied in a higher-order moment computation, and therefore can impact security in the noisy leakage model. This is in fact exactly the goal of the bounded moment model. Namely, it aims at simplifying security evaluations by splitting the tasks of evaluating the leakages' deterministic part (captured by their moments) and probabilistic part (aka noise). Concrete security against side-channel attacks is ensured by two ingredients: a high security order and sufficient noise.

4.3 Abstract implementation settings

In the following, we exploit our model in order to study the impact of parallelism in general terms. For this purpose, we follow the usual description of masked implementations as a sequence of leaking operations. Furthermore, and in order to first abstract away physical specificities, we consider so-called (noiseless) "abstract implementations", simplifying Equation (2) into:

$$L_c = \alpha_1 \cdot \mathsf{G}_1\left(\mathcal{Y}_c(1)\right) + \alpha_2 \cdot \mathsf{G}_2\left(\mathcal{Y}_c(2)\right) + \ldots + \alpha_{n_c} \cdot \mathsf{G}_{nc}\left(\mathcal{Y}_c(n_c)\right). \quad (3)$$

Such simplifications allow analyzing masked implementations independent of their concrete instantiation in order to detect algorithmic flaws. Note that having $R_c \neq 0$ cannot change conclusions regarding the security order of an implementation (in the bounded moment model), which is the only metric we consider in this paper. Indeed, this order only depends on the smallest key-dependent moment of the leakage distribution, which is independent of the additive noise. By contrast, the variance of R_c affects the concrete information leakage of an implementation. We recall that algorithmically sound masked implementations do not mandatorily lead to physically secure implementations (e.g. because of the independence issues discussed in Section 9 or a too low noise). Yet, and as mentioned in Section 3.2, testing the security of abstract implementations of masking schemes (in the probing or bounded moment models) is a useful preliminary, before performing expensive evaluations of concrete implementations.

5 Serial security implies parallel security

We now provide our first result in the bounded moment model. Namely, we establish an intuitive reduction between security of parallel implementations in the bounded moment model and security of serial implementations in the probing model. For this purpose, we also formalize our serial and parallel computation models. One useful and practical consequence of the reduction is that one can adapt existing tools for proving security in the bounded moment model, either by implementing a program transformation that turns parallel implementations into serial ones, or by adapting these tools to parallel implementations.

Intuition. In order to provide some intuition for the reduction, recall that the leakage samples of an abstract parallel implementation are of the form:

$$L_c = \mathcal{Z}_c(1) + \mathcal{Z}_c(2) + \ldots + \mathcal{Z}_c(n_c),$$

with $\mathcal{Z}_c(i) = \alpha_i \cdot \mathsf{G}_i(\mathcal{Z}_c(i))$, and that the bounded moments are of the form:

$$\mathsf{E}(L_1^{o_1} \times L_2^{o_2} \times \ldots \times L_r^{o_r}).$$

Therefore, by linearity of the expectation, mixed moments at order d are independent of secrets provided all quantities of the form:

$$\mathsf{E}\left((\mathcal{Z}_1(1))^{o_{1,1}} \times \ldots \times (\mathcal{Z}_1(n_1))^{o_{1,n_1}} \times (\mathcal{Z}_r(1))^{o_{r,1}} \times \ldots \times (\mathcal{Z}_r(n_r))^{o_{r,n_r}}\right), \quad (4)$$

are independent of secrets, for all $o_{1,1}, \ldots o_{r,n_r}$ whose sum is bounded by o. Note that there are at most o pairs (i, j) such that $o_{i,j} \neq 0$. Let $(i_1, n_1) \ldots (i_k, n_k)$ with $k \leq o$ be an enumeration of these pairs. Therefore, in order to establish that Equation (4) is independent of the secrets, it is sufficient to show that the tuple $\langle \mathcal{Z}_{i_1}(n_1), \ldots, \mathcal{Z}_{i_k}(n_k) \rangle$ is independent of these secrets. This in fact corresponds exactly to proving security in the probing model at order o.

Formalization. The theoretical setting for formalizing the reduction is a simple parallel programming language in which programs are sequences of basic instructions (note that adding for loops poses no further difficulty). A basic instruction is either a parallel assignment:

$$\langle a_1, \ldots, a_n \rangle := \langle e_1, \ldots, e_n \rangle,$$

where $e_1, \ldots, e_n$ are expressions built from variables, constants, and operators, or a parallel sampling:

$$\langle a_1, \ldots, a_n \rangle \leftarrow \langle \mu_1, \ldots, \mu_n \rangle,$$

where $\mu_1, \ldots, \mu_n$ are distributions. Despite its simplicity, this formalism is sufficient to analyse notions used for reasoning about threshold implementations, for instance non-completeness. More importantly, one can also define the notion of leakage associated to the execution of a program. Formally, an execution of a program c of length ℓ is a sequence of states $s_0 \ldots s_\ell$, where s_0 is the initial state and the state s_{i+1} is obtained from the state s_i as follows:

- If the ith-instruction is a parallel assignment, $\langle a_1, \ldots, a_n \rangle := \langle e_1, \ldots, e_n \rangle$ by evaluating the expressions $e_1 \ldots e_n$ in state s_i, leading to values $v_1 \ldots v_n$, and updating state s_i by assigning values $v_1 \ldots v_n$ to variables $a_1 \ldots a_n$;
- if the ith-instruction is a parallel sampling, $\langle a_1, \ldots, a_n \rangle \leftarrow \langle \mu_1, \ldots, \mu_n \rangle$ by sampling values $v_1 \ldots v_n$ from distributions $\mu_1 \ldots \mu_n$, and updating the state s_i by assigning the values $v_1 \ldots v_n$ to the variables $a_1 \ldots a_n$.

By assigning to each execution a probability (formally, this is the product of the probabilities of each random sampling), one obtains for every program c of length ℓ a sequence of distributions over states $\sigma_0 \sigma_1 \ldots \sigma_\ell$, where σ_0 is the distribution $\mathbb{1}_{s_0}$. The leakage of a program is then a sequence $L_1 \ \ldots \ L_\ell$, defined by computing for each i the sum of the values held by the variables assigned by the ith instruction, that is $a_1 + \ldots + a_n$ for parallel assignments (or samplings). The mixed moments at order o then simply follow Definition 1. As for the serial programming language, instructions are either assignments $a := e$ or sampling $a \leftarrow \mu$. The semantics of a program are defined similarly to the parallel case. Order o security of a serial program in the probing model amounts to show that each o-tuple of intermediate values is independent of the secret.

Without loss of generality, we can assume that parallel programs are written in static single assignment form, meaning that variables: *(i)* appear on the left hand side of an assignment or a sampling only once in the text of a program;

(ii) are defined before use (i.e. they occur on the left of an assignment or a sampling before they are used on the right of an assignment); *(iii)* do not occur simultaneously on the left and right hand sides of an assignment. Under such assumption, any serialization that transforms parallel assignments or parallel samplings into sequences of assignments or samplings preserve the semantics of programs. For instance, the left to right serialization transforms the parallel instructions $\langle a_1, \ldots, a_n \rangle := \langle e_1, \ldots, e_n \rangle$ and $\langle a_1, \ldots, a_n \rangle \leftarrow \langle \mu_1, \ldots, \mu_n \rangle$ into $a_1 := e_1; \ldots; a_n := e_n$ and $a_1 \leftarrow \mu_1; \ldots; a_n \leftarrow \mu_n$ respectively.

Reduction theorem. We can now state the reduction formally:

Theorem 1. *A parallel implementation is secure at order o in the bounded moment model if its serialization is secure at order o in the probing model.*

Proof. Assume that a parallel implementation is insecure in the bounded moment model but its serialization is secure in the probing model. Therefore, there exists a mixed moment:

$$\mathsf{E}(L_1^{o_1} \times L_2^{o_2} \times \ldots \times L_r^{o_r}),$$

that is dependent of the secrets. By definition of leakage vector, and properties of expectation, there exist program variables $a_1, \ldots, a_k$, with $k \leq o$, and $o'_1, \ldots, o'_k$ with $\sum_i o_i \leq o$ such that:

$$\mathsf{E}(a_1^{o'_1} \times a_2^{o'_2} \times \ldots \times a_k^{o_k})$$

is dependent of secrets, contradicting the fact (due to security of serialization in the probing model) that the tuple $\langle a_1, \ldots, a_k \rangle$ is independent of secrets. □

Note that concretely, this theorem suggests the possibility of efficient "combined security evaluations", starting with the use the formal verification tools to test probing security, and following with additional tests in the (weaker) bounded moment model in case of negative results (see the examples in Section 8).

Interestingly, it also backs up a result already used in [**?**] (Theorem 1), where the parallel nature of the implementations was not specifically discussed but typically corresponds to the experimental case study in this paper.

6 Parallel algorithms

In this section, we describe regular and parallelizable algorithms for secure (additively) masked computations. For this purpose, we denote a vector of d shares as $\boldsymbol{a} = [a_1, a_2, \ldots, a_d]$, the rotation of this vector by q positions as $\mathsf{rot}(\boldsymbol{a}, q)$, and the bitwise addition (XOR) and multiplication (AND) operations between two vectors as $\boldsymbol{a} \oplus \boldsymbol{b}$ and $\boldsymbol{a} \cdot \boldsymbol{b}$. For concreteness, our analyses focus on computations in $\mathsf{GF}(2)$, but their generalization to larger fields is straightforward.

6.1 Parallel refreshing

As a starting point, we exhibit a very simple refreshing algorithm that has constant time in the parallel implementation setting and only requires d bits of fresh uniform randomness. This refreshing is given in Algorithm 1 and an example of abstract implementation for $d = 3$ shares is in Figure 3, where we mark the load-

Algorithm 1 Parallel refreshing algorithm.

Input: Shares $\boldsymbol{a}$ satisfying $\bigoplus_i a_i = a$, uniformly random vector $\boldsymbol{r}$.
Output: Refreshed shares $\boldsymbol{b}$ satisfying $\bigoplus_i b_i = a$.

$\boldsymbol{b} = \boldsymbol{a} \oplus \boldsymbol{r} \oplus \mathsf{rot}(\boldsymbol{r}, 1)$;
return $\boldsymbol{b}$.

ing of input shares in gray, the loading and rotation of fresh randomness in green and the additions with this fresh randomness in blue. Such an implementation runs in 5 cycles (independent of d) and leads to 4 observable leakages (since the leakage of the random vector $\boldsymbol{r}$ and its rotation gives rise to the same sum).

cycles	1	2	3	4	5
shares	a_1 a_2 a_3	r_1 r_2 r_3	r_3 r_1 r_2	$x_1=a_1\oplus r_1$ $x_2=a_2\oplus r_2$ $x_3=a_3\oplus r_3$	$b_1=x_1\oplus r_3$ $b_2=x_2\oplus r_1$ $b_3=x_3\oplus r_2$
leakage	$\sum a_i$	$\sum r_i$	$\sum r_i$	$\sum x_i$	$\sum b_i$
	gray	green	green	blue	blue

Fig. 3. Abstract implementation of a 3-share refreshing.

6.2 Parallel multiplication

Next, we consider the more challenging case of parallel multiplication with the similar goal of producing a simple and systematic way to manipulate the shares and fresh randomness used in the masked computations. For this purpose, our starting observation is that existing secure (serial) multiplications such as [**?**] (that we will mimic) essentially work in two steps: first a product phase that computes a d^2-element matrix containing the pairwise multiplications of all the shares, second a compressing phase that reduces this d^2-element matrix to a d-element one (using fresh randomness). As a result, and given the share vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ of two sensitive values a and b, it is at least possible to perform each pair of cross products $a_i \cdot b_j$'s and $a_j \cdot b_i$'s with XOR and rotation operations, and without refreshing. By contrast, the direct products $a_i \cdot b_j$ have to be separated by fresh randomness (since otherwise it could lead to the manipulation of sensitive values during the compression phase, e.g. $(a_i \cdot b_i) \oplus (a_i \cdot b_j) = a_i \cdot (b_i \oplus b_j)$. A

similar reasoning holds with the uniform randomness used between the XORs of the compression phase. Namely, every fresh vector can be used twice (in its original form and rotated by one) without leaking additional information.

This rationale suggests a simple multiplication algorithm that has linear time complexity in the parallel implementation setting and requires $\lceil \frac{d-1}{4} \rceil$ random vectors of d bits (it can be viewed as an adaptation of the algorithms in [**?**]). We first highlight it based on its abstract implementation in Figure 4, which starts with the loading and rotation of the input shares (gray cycles), then performs the product phase (red cycles) and finally compresses its output by combining the addition of fresh randomness (blue cycles) and accumulation (orange cycles). In general, such an implementation runs in $< 5d$ cycles for d shares, with slight variations depending on the value of d. For $d \bmod 4 = 3$ (as in Figure 4) it is "complete" (i.e. ends with two accumulation cycles and one refreshing). But for $d \bmod 4 = 0$ it ends with a single accumulation cycle, for $d \bmod 4 = 1$ it ends with two accumulation cycles and for $d \bmod 4 = 2$ it ends with an accumulation cycle and a refrehing. An accurate description is given in Algorithm 3.

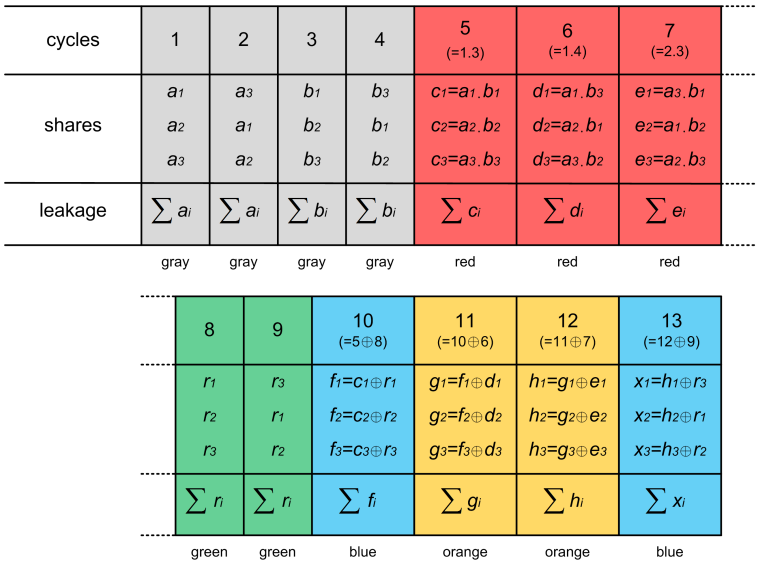


Fig. 4. Abstract implementation of a 3-share multiplication.

Impact for multivariate (aka horizontal) attacks. In simplified (asymptotic) terms, the security proofs for masked implementations in [**?**] state that the success rate of a side-channel attack can be bounded by $(\sigma_n^2)^{-d}$, with d the number of shares and σ_n^2 the noise affecting the leakage of each share, if $\sigma_n^2 \geq d$ (where the d factor is due to the computation of the partial products in the multiplication algorithm of [**?**]). In a recent work, Batistello et al. [**?**] showed that

since each share in a serial implementation is manipulated d times, it is possible to pre-process their leakages by averaging. This reduces their noise variance by an additional factor d, meaning that the noise condition becomes $\sigma_n^2 \geq d^2$ overall [**?**]. Interestingly, while such a pre-processing is also possible in our parallel case, due to the identical grey cycles of Figure 4, the number of times each sum of shares is manipulated is now bounded to 2 (independent of d) which mitigates the impact of such attacks. That is, we can maintain the noise condition for secure masking at $\sigma_n^2 \geq 2d$ (where the factor 2 comes from the repetition of the grey cycles), even when exploiting the averaging pre-processing.

7 Case studies

By Theorem 1, security in the bounded moment model of a parallel implementation can be established from security of its serialization in the probing model. Therefore, it is possible to use existing formal methods to test the security of parallel implementations, by first pre-processing them into a serial ones, and feeding the resulting serial programs into a verification tool. In this section, we report on the successful automated analysis of several parallel implementations, including the parallel refreshing and multiplication presented in the previous section, and serial composition of parallel S-boxes. Note that, due to the algorithmic complexity of the verification task, we only establish security at small orders. However, we also note that, although our main design constraint was for our algorithms to be easily implementable in parallel, the use of automated tools – as opposed to manual analyses – to verify their security has yielded algorithms that match or improve on the state-of-the-art in their randomness requirements at these orders. All experiments reported in this section are based on the current version of the tool of [**?**]. This version supports automated verification of two properties: the usual notion of probing security, and a strictly stronger notion, recently introduced in [**?**] under the name *strong non-interference* (SNI), which is better suited to the compositional verification of large circuits.

7.1 Parallel refreshing

We first consider the parallel refreshing algorithm from the previous section.

Theorem 2 (Security of Algorithm 1). *The refreshing in Algorithm 1 is secure at order $d-1$ in the bounded moment model for all $d \leq 7$.*

By Theorem 1, it is sufficient to prove $(d-1)$-probing security to get security at order $d-1$ in the bounded moment model. We do so using the tool by Barthe et al. [**?**] for each order $d \leq 7$. Table 1 shows the verification time for each proof.

In addition, we consider the problem of how to construct a SNI mask refreshing gadget that behaves as well with respect to our parallel computation model. We rely on the current version of the tool from Barthe et al. [**?**], which supports the verification of strong non-interference properties. Table 2 reports the verification results for some number of mask refreshing algorithms, constructed simply

d	$(d-1)$-b.m.	time (s)
3	✓	1
4	✓	1
5	✓	2
6	✓	20
7	✓	420

Table 1. Probing and bounded moment security of Algorithm 1.

by iterating Algorithm 1 (denoted R_d). We denote with R_d^n the algorithm that iterates R_d n times. Table 2 also shows the randomness requirements both for our algorithm and for the only other known SNI mask refreshing gadget, based on Ishai, Sahai and Wagner's multiplication algorithm [**?**].

Alg.	d	$(d-1)$-SNI	# rand. bits		time (s)
			our alg.	[**?**]	
R_d	3	✓	3	3	1
R_d	4	✓	4	6	1
R_d	5	✗	5	10	1
R_d^2	5	✓	10	10	1
R_d^2	6	✓	12	15	1
R_d^2	7	✓	14	21	1
R_d^2	8	✗	16	28	1
R_d^3	8	✓	24	28	4
R_d^3	9	✓	27	36	36
R_d^3	10	✓	30	45	288
R_d^4	11	✓	40	55	3045

Table 2. SNI secure variants of Algorithm 1.

These experiments show that, for small masking orders, there exist regular mask refreshing gadgets that are easily parallelizable, suitable for the construction of secure circuits by composition, and that have small randomness requirements. This fact is particularly useful when viewed through the lens of Theorem 1. Indeed, SNI gadgets are instrumental in easily proving probing security for large circuits [**?**], which Theorem 1 then lifts to the bounded moment model and parallel implementations. We conjecture that iterating the simple mask refreshing gadget from Algorithm 1 $\lceil (d-1)/3 \rceil$ times always yields a $(d-1)$-SNI mask refreshing algorithm over d shares. The resulting algorithm is easily parallelizable and requires $\lceil (d-1)/3 \rceil \cdot d$ bits of randomness (marginally improving on the $d \cdot (d-1)/2$ bits of randomness from the ISW-based mask refreshing). We leave a proof of strong non-interference for all d's as future work.

7.2 Parallel multiplication

We now consider the parallel multiplication algorithm from the previous section (Algorithm 3), and prove its security for small orders.

Theorem 3 (Security of Algorithm 3). *The multiplication in Algorithm 3 is secure at order $d-1$ in the bounded moment model for all $d \leq 7$.*

By Theorem 1, it is sufficient to prove $(d-1)$-probing security to get security at order $d-1$ in the bounded moment model. We do so using the tool by Barthe et al. [**?**] for each $d \leq 7$. Table 3 shows the verification time for each instance.

		# rand. bits		
d	$(d-1)$-b.m.	our alg.	[**?**]	time (s)
3	✓	3	2	1
4	✓	4	4	1
5	✓	5	5	2
6	✓	12	11	17
7	✓	14	15	480

Table 3. Probing and bounded moment security of Algorithm 3.

We also show a comparison of the randomness requirement of our algorithm and those of Belaïd et al. [**?**]. Note that we sometimes need one additional random bit compared to the algorithm of Belaïd et al. [**?**]. This is due to our parallelization constraint: instead of sampling uniform sharings of 0, we only allow ourselves to sample uniformly random vectors and to rotate them.

As before, we now investigate some combinations of Algorithms 1 and 3 in the hope of identifying regular and easily parallelizable SNI multiplication algorithms. The results of the experiments are shown in Table 4, where $\odot_d$ is Algorithm 3, specialized to d shares. In addition to showing whether or not the algorithm considered is SNI, the table shows verification times and compares the randomness requirements of our algorithm with that of the multiplication algorithm by Ishai, Sahai and Wagner, which is the best known SNI multiplication algorithm in terms of randomness. As with the original tool by Barthe et al. [**?**], the verification task is constrained to security orders $o \leq 8$ for circuits involving single multiplications due to the exponential nature of the problem it tackles. We conjecture that the combination of our multiplication algorithm with a single

			#rand. bits		
Algorithm	d	$(d-1)$-SNI	our alg.	[**?**]	time (s)
$\odot_d$	3	✓	3	3	1
$\odot_d$	$d \geq 4$	✗	$d(d-1)/4$	$d(d-1)/2$	-
$R_d \circ \odot_d$	4	✓	8	6	1
$R_d \circ \odot_d$	5	✓	10	10	1
$R_d \circ \odot_d$	6	✓	18	15	39
$R_d \circ \odot_d$	7	✓	21	21	2647
$R_d \circ \odot_d$	8	✓	24	28	176400

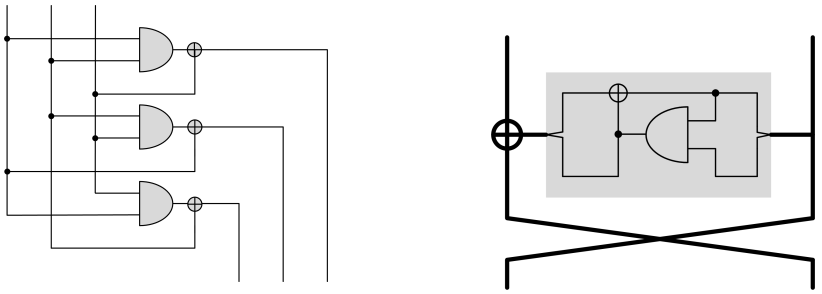
Table 4. SNI security for variants of Algorithm 3.

refreshing is SNI for any d. This is intuitively justified by the fact our multiplication algorithm includes a number of "half refreshings", which must be combined

with a final refreshing for the d's such that the it ends with an accumulation step. We leave the proof of this conjecture as an interesting open problem.

7.3 S-boxes and Feistel networks

In order to better investigate the effects on the security of larger circuits of reducing the randomness requirements of the multiplication and refreshing algorithms, we now consider small S-boxes, shown in Figure 5, and their iterations. Figure 5(a) describes a simple 3-bit S-box similar to the "Class 13" S-box of



(a) 3-bit S-box (sbox_3) (b) 4-bit S-box (sbox_4)

Fig. 5. Examples of elementary circuits.

Ullrich et al. [**?**]. Figure 5(b) describes a 4-bit S-box constructed by applying a Feistel construction to a 2-bit function. Table 5 shows verification results for iterations of these circuits for several small orders, exhibiting some interesting compositional properties for these orders. sbox_3 denotes the circuit from Figure 5(a), sbox_4 denotes the circuit from Figure 5(b), and sboxr_4 denotes the circuit from Figure 5(b), modified so that the upper output of its inner transformation is refreshed. As before, integer exponents denote sequential iteration. We note that, although there is no evidence that iterating sbox_4 longer yields

$d = 3$

Algorithm	2-b.m.	time (s)
sbox_3	✓	1
sbox_3^2	✓	1
sbox_4	✓	1
sbox_4^2	✓	1
sbox_4^3	✓	714
sboxr_4^3	✓	1
sboxr_4^4	✓	3
sboxr_4^5	✓	7
sboxr_4^6	✓	12

$d = 4$

Algorithm	3-b.m.	time (s)
sbox_3	✓	13
sbox_3^2	✓	322
sbox_4	✓	2
sbox_4^2	✓	67

Table 5. Probing and bounded moment security of small S-boxes.

insecure circuits, obtaining convincing security results for more than 3 iterations using automated tools seems unfeasible without relying on compositional principles. In particular, inserting a single mask refreshing operation per Feistel round greatly speeds up the verification of large iterations of the 4-bit S-box from Figure 5(b). This highlights possible interactions between tools oriented towards the verification of small optimized circuits for particular values of d [**?**,**?**,**?**] and tools geared towards the more efficient but less precise verification of large circuits [**?**]. The ability to make our algorithms SNI allows us to directly take advantage of this "randomness complexity vs. verification time" tradeoff.

8 Separation results

The previous sections illustrated that the reduction from security in the bounded moment model for parallel implementations to security in the probing model for their corresponding serialized implementations gives solutions to a number of technical challenges in the design of secure masking schemes. We now question whether the weaker condition required for security in the bounded moment model allows some implementations to be secure in this model and not in the probing model. We answer this question positively, starting with somewhat specialized but illustrative examples, and then putting forward a practically relevant separation between these models in the context of continuous leakages.

8.1 Specialized encodings and masking schemes

Starting example. Let us imagine a 2-cycle parallel implementation manipulating two shares in each cycle. In the first cycle, the same random bit r is loaded twice, giving rise to a state (r, r). In the second cycle, a shared sensitive value a is loaded twice, giving rise to a state $(a \oplus r, \overline{a} \oplus r)$. Clearly, in the probing model two probes (on r and $a \oplus r$) are sufficient to learn a. But for an adversary observing the abstract leakages of this parallel implementations (i.e. the arithmetic sum for each cycle), and for a particular type of leakage function such that $\alpha_i^j = 1$ and $\mathsf{G}_i^j = \mathsf{Id}$ in Equation (2), the first cycle will only reveal $r + r$ while the second cycle will reveal a constant 1. So no combinations of these leakages can be used to recover a. An even simpler example would be the parallel manipulation of a and $\overline{a}$ which trivially does not leak any information if their values are just summed. Such implementations are known under the name "dual-rail pre-charged" implementations in the literature [**?**]. Their main problem is that they require much stronger physical assumptions than masked implementations. That is, the leakages on the shares a and $\overline{a}$ do not only need to be independent but identical, which turns our to be much harder to achieve in practice [**?**].

Leakage squeezing and low entropy masking schemes. Interestingly, the literature provides additional examples of countermeasures where the security order is larger in the bounded moment model than in the probing model. In particular, leakage squeezing and low entropy masking schemes exploit special types of encodings such that the lowest key-dependent statistical moment of

their leakage distrubutions is larger than the number of shares, if the leakage function's deterministic part is linear [**?**,**?**], i.e. if $\mathsf{G}_i^j = \mathsf{Id}$ in Equation (2). Note that this requirement should not be confused with the global linearity requirement of Equation (2). That is, what masking generally requires to be secure is that the different shares are combined linearly (i.e. that Equation (2) is a first-degree polynomial of the $\mathsf{G}_i^j(\mathcal{Y}_i(j))$'s). Leakage squeezing and low entropy masking schemes additionally require that the (local) G_i^j functions are linear.

The previous examples show that in theory, there exist leakage functions such that the security order in the bounded moment model is higher than the security order in the probing model, which is sufficient to prove separation. Yet, as previously mentioned, in practice the identical (resp. linear) leakage assumption required for dual-rail pre-charged implementations (resp. leakage squeezing and low entropy masking schemes) is extremely hard to fulfill (resp. has not been thoroughly studied yet). So this is not a general separation for any implementation. We next present such a more general separation.

8.2 The continuous leakage separation

A continuous probing attack against the refreshing of Algorithm 1. Up to this point of the paper, our analyses have considered "one-shot" attacks and security. Yet, in practice, the most realistic leakage models consider adversaries who can continuously observe several executions of the target algorithms. Indeed, this typically corresponds to the standard DPA setting where senstive information is extracted by combining observations from many successive runs [**?**]. Such a setting is reflected in the continuous t-probing model of Ishai, Sahai and Wagner [**?**], where the adversary can learn t intermediate values produced during the computation of each execution of the algorithm. It implies that over time the adversary may learn much more information than just the t values – and in particular more than d, the number of shares. To be concrete, in a continuous attack that runs for q executions the adversary can learn up to tq intermediate values, evenly distributed between the executions of the algorithm.

Designing strong mask refreshing schemes that achieve security in the continuous t-probing model is a non-trivial task. In this section, we show that Algorithm 1 can be broken for any number of shares d, if the refreshing is repeated consecutively for d times and in each execution the adversary can learn up to 3 intermediate values. To explain the attack, we first generalize this algorithm to d executions, with $a_1^{(0)}, \ldots, a_d^{(0)}$ the initial encoding of some secret bit a, as given in Algorithm 2. The Lemma below gives the attack. Similar attacks are used in [**?**] for the inner product masking in the bounded leakage model.

Lemma 2. *Let a be a uniformly chosen secret bit, $d \in \mathbb{N}$ a number of shares and consider Algorithm 2. In each iteration of the* for *loop there exists a set of* 3 *probes such that after d iterations the secret a can be learned.*

Proof. We show that, if the adversary can probe 3 intermediate values in each iteration of the parallel refreshing for d iterations, then he can recover the secret

Algorithm 2 d-times execution of the parallel refreshing algorithm.

Input: Shares $\boldsymbol{a}^{(0)}$ satisfying $\bigoplus_i a_i^{(0)} = a$ and
d random vectors $\boldsymbol{r}^{(i)}$.
Output: Refreshed shares $\boldsymbol{a}^{(d)}$ satisfying $\bigoplus_i a_i^{(d)} = a$.

```
for i = 1 to d do
  a^(i) = a^(i-1) ⊕ r^(i) ⊕ rot(r^(i), 1);
end for
return a^(d).
```

bit a. The proof is by induction, where we show that, after learning the values of his 3 probes in the ith iteration, the adversary knows the sum of the first i shares of a, that is $A_1^i := \bigoplus_{j=1}^{i} a_j^{(i)}$. Since $A_1^d := \bigoplus_{j=1}^{d} a_j^{(d)} = a$, after d iterations, the adversary thus knows the value of a. In the first iteration, a single probe on share $a_1^{(1)}$ is sufficient to learn $A_1^1 := a_1^{(1)}$. We now prove the inductive step. Let $1 < \ell \leq d$. Suppose after the $(\ell-1)$th execution, we know: $A_1^{\ell-1} := \bigoplus_{j=1}^{\ell-1} a_j^{(\ell-1)}$. In the ℓth iteration, the adversary probes $r_d^{(\ell)}, r_{\ell-1}^{(\ell)}$ and $a_\ell^{(\ell)}$, allowing him to compute A_1^ℓ using the following equalities:

$$A_1^\ell = \bigoplus_{j=1}^{\ell} a_j^{(\ell)} = a_\ell^{(\ell)} \oplus \bigoplus_{j=1}^{\ell-1} a_j^{(\ell)} = a_\ell^{(\ell)} \oplus \bigoplus_{j=1}^{\ell-1} a_j^{(\ell-1)} \oplus r_j^{(\ell)} \oplus r_{j-1}^{(\ell)}$$
$$= a_\ell^{(\ell)} \oplus r_d^{(\ell)} \oplus r_{\ell-1}^{(\ell)} \oplus \bigoplus_{j=1}^{\ell-1} a_j^{(\ell-1)} = a_\ell^{(\ell)} \oplus r_d^{(\ell)} \oplus r_{\ell-1}^{(\ell)} \oplus A_1^{\ell-1},$$

where we use the convention that for any j we have $r_0^{(j))} = r_d^{(j))}$. Since all values after the last equality either are known from the previous round or have been learned in the current round the above concludes the proof. □

Continuous security of Algorithm 1 in the bounded moment model. The previous attack crucially relies on the fact that the adversary can move his probes adaptively between different iterations, i.e. in the ith execution he must learn different values than in the $(i-1)$th execution. This implies that in practice he would need to exploit jointly $\approx 3d$ *different* time samples from the power trace. We now show that such an attack is not possible in the (continuous) bounded moment model. The only difference between the continuous bounded moment model and the one-shot bounded moment model is that the first offers more choice for combining leakages as there are q-times more cycles. More precisely, the natural extension of bounded moment security towards a continuous setting requires that the expectation of any oth-degree polynomial of leakage samples among the q leakage vectors that can be observed by the adversary is independent of any sensitive variable $Y \in \{0, 1\}$ that is produced during the q

executions of the implementation. Thanks to Lemma 1, we know that a sufficient condition for this condition to hold is that is that the expectation of all the monomials is independent of Y. So concretely, we only need that for any tuple of o possible clock cycles $c_1, c_2 \ldots, c_o \in [1, qN]$, we have:

$$\Pr[Y = 0] = \Pr[Y = 0 | \mathsf{E}[L_{c_1} \times L_{c_2} \times \ldots \times L_{c_o}]],$$
$$\Pr[Y = 1] = \Pr[Y = 1 | \mathsf{E}[L_{c_1} \times L_{c_2} \times \ldots \times L_{c_o}]].$$

In the one-shot bounded moment model $c_1, c_2 \ldots, c_o$ would only run in $[1, N]$. Our following separation result additionally needs a specialization to stateless primitives. By stateless, we mean primitives such as block ciphers that only need to maintain a constant secret key in memory from one execution to the other.

Theorem 4. *The implementation of a stateless primitive where the secret key is refreshed using Algorithm 1 is secure at order o in the continuous bounded moment model if it is secure at order o in the one-shot probing model.*

Proof (sketch). We consider an algorithm for which a single execution takes N cycles which is repeated q times. We can view the q-times execution of the algorithm as a computation running for qN cycles. Since we are only interested in protecting stateless primitives, individual executions are only connected via their refreshed key. Hence, the q-times execution of the N-cycle implementation can be viewed as a circuit consisting of q refreshings of the secret key using Algorithm 1, where each refreshed key is used as input for the stateless masked implementation. If we show that this "inflated" circuit is secure against an adversary placing up to o probes in these qN cycles (in total and not per execution as in the continuous probing model), the result follows by Theorem 1.

For this purpose, we first observee that o probes just in the part belonging to the q-times refreshing do not allow the adversary to learn the masked secret key. This follows from the fact that probing o values in a one-shot execution of the refreshing (Algorithm 1) does not allow the adversary to learn this masked secret key. More precisely, any such probes in the refreshing can be directly translated into probes on the initial encoding (and giving the appropriate randomness of the refreshing to the adversary for free). This means that any probe in the refreshing part allows to learn at most a single share of the masked secret key going into the stateless masked implementation. Moreover, we know by assumption that a single-shot execution of the implementation is o-probing secure. This implies that even after o probes inside the masked implementation there still must exist one share of the masked state of which these probes are independent. More generally, placing $o - i$ probes in the masked implementation must imply that these probes are independent of at least $i + 1$ shares of the masked state, since otherwise the remaining i probes can be placed at the unknown input shares to get a correlation with the masked secret key. As a result, we can also reveal all of the shares of the input encoding except for these $i + 1$ shares that are independent. Therefore, by simply adding up the probes, we get that even placing o probes inside of the inflated circuit maintains security. □

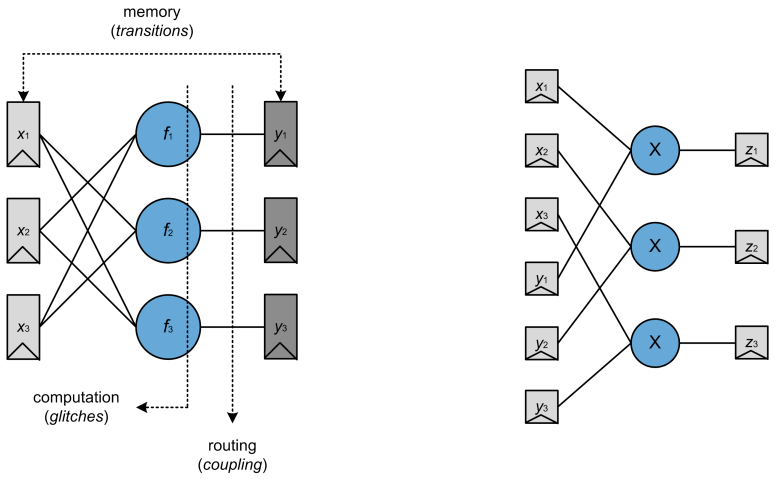
Note that the above argument with the inflated circuit and the special use of the refreshing fails to work when we consider stateful primitives. In such a setting, the refreshing may interact with other parts of the circuit. Hence, we would need a stronger (composable) refreshing to achieve security in this case, in order to deal with the fact that Algorithm 1 could then appear at arbitrary positions in the computation. As already mentioned, the security condition of the bounded moment model is significantly weaker than in the probing model, which is what allows us to reach this positive result. Intuitively, security in the probing model requires that, given a certain number of probes, no information is leaked. By contrast, security in the bounded moment model only requires that this information is hard to exploit, which is captured by the fact that the lowest informative statistical moment in the leakage distribution observed by the adversary is bounded. This model nicely captures the reality of concrete side-channel attacks, where all the points of a leakage traces (as in Figure 1) are available to this adversary, and we want to ensure that he will at least have to estimate a higher-order moment of this leakage distribution in order to extract sensitive information (a task that is exponentially hard in o if the distribution is sufficiently noisy). We believe this last result is particularly relevant for cryptographic engineers, since it clarifies a long standing gap between the theory and practice of masking schemes regarding the need of complex refreshing schemes. Namely, we are able to show that simple refreshing schemes such as in Section 6.1 indeed bring sufficient security against concrete higher-order side-channel attacks.

Note also that it is an interesting open problem to investigate the security of our simple refreshing scheme in the continuous noisy leakage model. Intuitively, extending the attack of Lemma 2 to this setting seems difficult. Take the second step for example: we have learned A_1^1 and want to learn A_1^2 with three noisy probes. If the noise is such that we do not learn A_1^2 exactly, then observing again threee probes with an independent noise will not help much (since we cannot easily combine the information on the fresh A_1^2, and would need to collect information on all d shares to accumulate information on the constant secret). As for Theorem 1 in Section 5, we can anyway note that the bounded moment model allows obtaining much easier connections to the (more theoretical) probing model than the (more general but more involved) noisy leakage model.

9 Independence issues

Before concluding, we discuss one important advantage of threshold implementation for hardware (parallel) implementations, namely their better resistance against glitches. We take advantage of and generalize this discussion to clarify the different independence issues that can affect leaking implementations, and detail how they can be addressed in order to obtain actual implementations that deliver the security levels guaranteed by masking security proofs.

Implementation defaults. As a starting point, we reproduce a standard example of threshold implementation, in Figure 6(a), which corresponds to the secure execution of a small Boolean function $f(x)$, where both the function



(a) Threshold implementation. (b) 3-bit partial product.

Fig. 6. Independence issues and threshold implementations.

and the inputs/outputs are shared in three pieces. In this figure, the (light and dark) gray rectangles correspond to registers, and the blue circles correspond to combinatorial circuits. From this example, we can list three different types of non-independence issues that can occur in practice:

1. *Computational re-combining (or glitches).* In this first case, transient intermediate computations are such that the combinatorial part of the circuit re-combines the shares. This effect has been frequently observed in the literature under the name "glitches", and has been exploited to break (i.e. reduce the security order) of many hardware implementations (e.g. [**?**]).

2. *Memory re-combining (or transitions).* In this second case, non independence comes from register re-use and the fact that actual leakage may be proportional to the transition between the register states. For example, this would happen in Figure 6(a), if registers x_1 and y_1 (which depends on x_2, x_3) are the same. This effect has been frequently observed in the literature too, under the name "distance-based" or "transition-based" leakages, and has been exploited to break software implementations (e.g. [**?**,**?**]).

3. *Routing re-combining (or coupling).* In this final case, the re-combining is based on the physical proximity of the wires. The leakage function would then be proportional to some function of these wires. Such effects, known under the name "coupling", could break the additive model of Equation (2) in case of complex (e.g. quadratic) function. To the best of our knowledge, they have not yet been exploited in a concrete (published) attack.

Glitches, threshold implementations and non-completeness. One important contribution of threshold implementations is to introduce a sound algorithmic way to deal with glitches. For this purpose, they require their implementations to satisfy the "non-completeness" property, which requires (at order o) that any combination of up to o component functions f_i must be independent of at least one input share [**?**]. Interestingly, and as depicted in Figure 6(b), this property is inherently satisfied by our parallel multiplication algorithm, which is in line with the previous observations in [**?**] and the standard method to synthesize threshold implementations, which is based on a decomposition in quadratic functions [**?**]. Note that threshold implementations crucially rely on the separation of the non-complete f_i functions by registers. So in order to obtain both efficient and glitch-free implementations of Algorithm 3, it is typically advisable to implement it in larger fields (e.g. by extending our multiplication algorithm in $\mathsf{GF}(2^8)$ as for the AES) or to exploit parallelism via bitslicing [**?**].

Transition-based leakage. Various design solutions exist for this purpose. The straighforward one is simply to ensure that all the registers in the implementation are different, or to double the order of the masking scheme [**?**]. But this is of course suboptimal (since not all transitions are leaking sensitive information). So a better solution is to include transition-based leakages in the evaluation of masked implementations, a task which also benefits from the tools in [**?**].

Couplings. This last effect being essentially physical, there are no algorithmic/software methods to prevent it. Couplings are especially critical in the context of parallel implementation since the non-linearity they imply may break the the independent leakage assumption. (By contrast, in serial implementations this assumption is rather fulfilled by manipulating the shares at different cycles). So the fact that routing-based recombinations do not occur in parallel masked implementations is essentially an assumption that all designers have to make. In this respect, we note that experimental results of attacks against threshold implementations where several shares are manipulated in parallel (e.g. the ones listed in Section 4.1) suggest that this assumption is indeed well respected for current technologies. Yet, we also note that the risk of couplings increases with technology scaling [**?**]. Hence, in the latter case it is anyway a good design strategy to manipulate shares in larger fields, or to ensure a sufficient physical distance between them if masking is implemented in a bitslice fashion.

10 Conclusions and open problems

Parallel masked implementations are appealing in practice, since they mitigate the large cycle counts of masked software implementations. This paper introduces sound tools for reasoning about such parallel implementations and their security against side-channel attacks, and show how these tools can be used to recycle in a hardware setting the large amount of theoretical and practical work done in the context of software-oriented masking. Moreover, the paper shows that our model supports secure and efficient refreshing and multiplication algorithms.

These results lead to two important tracks for further research.

First, the bounded moment model that we introduce can be seen as an intermediate path between the conceptually simple probing model and the practically relevant noisy leakage model. As discussed in Section 8 (and illustrated in Figure 7), the bounded moment leakage model is strictly weaker than the probing model. Hence, it would be interesting to investigate whether bounded moment security implies noisy leakage security for certain classes of leakage functions. Clearly, this cannot hold in general since there exist different distributions with identical moments. Yet, and in view of the efficiency gains provided by moment-based security evaluations, it is an interesting open problem to identify the contexts in which this approach is sufficient, i.e. to find out when a leakage distribution is well enough represented by its moments. Building on and formalizing the results in [**?**] is an interesting direction for this purpose.

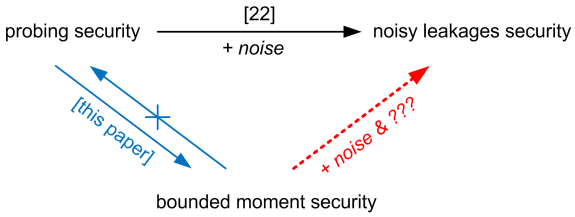


Fig. 7. Reductions between leakage security models.

Second, whenever discovering a bias in a masked implementation, our tools not only output the computation leading to this bias, but also its (possibly small) amplitude. Hence, the bounded moment model has great potential to extend the quantitative analysis in [**?**] (so far limited to first-order leakages) to the higher-order case. Relying on the fact that the biases may be quantitatively hard to exploit could lead to further reductions of the randomness requirements in masked implementations, e.g. by combining the evaluation of these biases with tools to analyze non-independent leakages introduced in [**?**] (Section 4.2).

Acknowledgements. Sebastian Faust is funded by the Emmy Noether Program FA 1320/1-1 of the German Research Foundation (DFG). François-Xavier Standaert is a research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in parts by projects S2013/ICE-2731 N-GREENS Software-CM, ONR Grants N000141210914 and N000141512750, FP7 Marie Curie Actions-COFUND 291803 and ERC project 280141.

Algorithm 3 Parallel multiplication algorithm.

Input: Shares $\boldsymbol{a}$ and $\boldsymbol{b}$ satisfying $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$, and $\lceil \frac{d-1}{4} \rceil$ uniformly random vectors $\boldsymbol{r}_i$.
Output: Shares $\boldsymbol{x}$ satisfying $\bigoplus_i a_i = a \cdot b$.

$\boldsymbol{c}_1 = \boldsymbol{a} \cdot \boldsymbol{b}$;
for $i = 1$ to $\left\lfloor \frac{d-[(d-3) \bmod 4]}{2} \right\rfloor$ **do**
  $\boldsymbol{c}_{2i} = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, i)$;
  $\boldsymbol{c}_{2i+1} = \mathsf{rot}(\boldsymbol{a}, i) \cdot \boldsymbol{b}$;
end for
if $d \bmod 4 = 0$ **then**
  $\boldsymbol{c}_d = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, d/2)$;
end if
if $d \bmod 4 = 1$ **then**
  $\boldsymbol{c}_{d-1} = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, \lfloor \frac{d}{2} \rfloor)$;
  $\boldsymbol{c}_d = \mathsf{rot}(\boldsymbol{a}, \lfloor \frac{d}{2} \rfloor) \cdot \boldsymbol{b}$;
end if
if $d \bmod 4 = 2$ **then**
  $\boldsymbol{c}_{d-2} = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, \lfloor \frac{d}{2} \rfloor)$;
  $\boldsymbol{c}_{d-1} = \mathsf{rot}(\boldsymbol{a}, \lfloor \frac{d}{2} \rfloor) \cdot \boldsymbol{b}$;
  $\boldsymbol{c}_d = \boldsymbol{a} \cdot \mathsf{rot}(\boldsymbol{b}, \lfloor \frac{d}{2} \rfloor + 1)$;
end if
$\boldsymbol{d}_1 = \boldsymbol{c}_1 \oplus \boldsymbol{r}_1$;
for $i = 1$ to $\left\lfloor \frac{d-[(d-3) \bmod 4]}{2} \right\rfloor$ **do**
  $\boldsymbol{d}_{3i-1} = \boldsymbol{d}_{3i-2} \oplus \boldsymbol{c}_{2i}$;
  $\boldsymbol{d}_{3i} = \boldsymbol{d}_{3i-1} \oplus \boldsymbol{c}_{2i+1}$;
  $\boldsymbol{d}_{3i+1} = \boldsymbol{d}_{3i} \oplus \mathsf{rot}(\boldsymbol{r}_{\lceil (i+1)/2 \rceil}, i \bmod 2)$;
end for
if $d \bmod 4 = 3$ **then**
  $\boldsymbol{x} = \boldsymbol{d}_{3\lfloor \frac{d}{2} \rfloor + 1}$
end if
if $d \bmod 4 = 0$ **then**
  $\boldsymbol{x} = \boldsymbol{d}_{3\lfloor \frac{d-1}{2} \rfloor + 1} \oplus \boldsymbol{c}_d$;
end if
if $d \bmod 4 = 1$ **then**
  $\boldsymbol{x} = \boldsymbol{d}_{3\lfloor \frac{d-2}{2} \rfloor + 1} \oplus \boldsymbol{c}_{d-1} \oplus \boldsymbol{c}_d$;
end if
if $d \bmod 4 = 2$ **then**
  $\boldsymbol{x} = \boldsymbol{d}_{3\lfloor \frac{d-3}{2} \rfloor + 1} \oplus \boldsymbol{c}_{d-2} \oplus \boldsymbol{c}_{d-1} \oplus \mathsf{rot}(\boldsymbol{r}_{\lceil \frac{d-1}{4} \rceil}, 1) \oplus \boldsymbol{c}_d$;
end if
return $\boldsymbol{x}$.