# Extractors Against Side-Channel Attacks: Weak or Strong?

Marcel Medwed[⋆], François-Xavier Standaert[⋆⋆].

UCL Crypto Group, Université catholique de Louvain.
Place du Levant 3, B-1348, Louvain-la-Neuve, Belgium.

**Abstract.** Randomness extractors are important tools in cryptography. Their goal is to compress a high-entropy source into a more uniform output. Beyond their theoretical interest, they have recently gained attention because of their use in the design and proof of leakage-resilient primitives, such as stream ciphers and pseudorandom functions. However, for these proofs of leakage resilience to be meaningful in practice, it is important to instantiate and implement the components they are based on. In this context, while numerous works have investigated the implementation properties of block ciphers such as the AES Rijndael, very little is known about the application of side-channel attacks against extractor implementations. In order to close this gap, this paper instantiates a low-cost hardware extractor and analyzes it both from a performance and from a side-channel security point of view. Our investigations lead to contrasted conclusions. On the one hand, extractors can be efficiently implemented and protected with masking. On the other hand, they provide adversaries with many more exploitable leakage samples than, e.g. block ciphers. As a result, they can ensure high security margins against standard (non-profiled) side-channel attacks and turn out to be much weaker against profiled attacks. From a methodological point of view, our analysis consequently raises the question of which attack strategies should be considered in security evaluations.

## 1 Introduction

Randomness extractors have recently been used as components of leakage-resilient cryptographic primitives such as stream ciphers [3, 19], pseudorandom functions [2, 16] and signatures [4]. They are also important in the design of public-key cryptosystems resistant to key leakage [12]. In this setting, the proofs of leakage-resilience usually rely on the fact that the amount of information leakage that is provided by one iteration of the extractor (i.e. when executed on one input) is bounded in some sense. As a result, an important requirement for these proofs to be meaningful in practice is that such a bounded leakage can actually be guaranteed by hardware designers. For this purpose, a first implementation and side-channel analysis of such a primitive was described in [14]. This work

[⋆] Postdoctoral researcher funded by the 7th framework European project TAMPRES.
[⋆⋆] Associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

analyzed an unprotected software implementation of an extractor. It was shown that, if no attention is paid, the extractor can actually lead to larger information leakages than an unprotected implementation of the AES Rijndael. Mainly, this happens because the extractor allows exploiting multiple leakage samples per plaintext. Thus, this previous work emphasized the importance of including the instantiation of cryptographic primitives in models of leakage resilience.

In this paper, we extend these preliminary investigations in two directions. First, we analyze a low-complexity extractor implemented in hardware (rather than software), and investigate the tradeoffs that such a design allows. Appealing design goals for the hardware implementation are a higher throughput and a leakage reduction due to parallelization. Second, we evaluate the impact of the masking countermeasure on the security of this extractor implementation. In particular, we exhibit an interesting homomorphic property that can be exploited to mask our design efficiently. The results of our hardware design-space evaluation show that the extractor can be masked up to unusually high orders while showing similar performance as a first-order masked block cipher implementation. As for the side-channel analysis results, they confirm part of the previous evaluations, showing that multi-sample per input attacks allow very efficient profiled side-channel attacks. Hence, depending on the adversarial strategies considered in the security evaluations, the implementation of a masked extractor may appear as weaker or stronger than the one of a block cipher. Positively, we show that hardware implementations of randomness extractors can guarantee a bounded leakage for bounded number of measurements. This validates their use as possible components of leakage-resilient constructions. Eventually, this work questions the methodologies for the evaluation of leaking devices in general, and underlines the large difference between profiled and non-profiled attacks that occurs for the extractor case.

The remainder of the paper is structured as follows. Sections 2 and 3 describe the analyzed low-complexity Hadamard extractor and its different hardware implementations. The side-channel attack scenario is detailed in Section 4. This is followed by an information theoretic analysis and security analysis in Sections 5 and 6. Finally, we draw conclusions in Section 7.

## 2 Low complexity extractor

In this section, we specify the instance of the low complexity Hadamard extractor, denoted as $\boxplus$, the implementation of which will be investigated in the remaining of the paper. It relies on the LFSR-based hashing technique from [8]. In order to compute $\boldsymbol{k}\boxplus\boldsymbol{x}$, one first expands $\boldsymbol{x} = s_0 s_1 \cdots s_{n-1}$ with the recurrence:

$$s_{i+n} \overset{def}{=} a_0 s_i + a_1 s_{i+1} + \cdots + a_{n-1} s_{i+n-1} \pmod 2, \qquad (1)$$

where the public constants $a_{n-1} \cdots a_0$ are the coefficients of a primitive polynomial of degree $n$. Then, we simply compute $m$ inner products (mod 2) between

$\boldsymbol{k}$ and the rows of a matrix filled with the expanded $\boldsymbol{x}$:

$$\boxplus : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m \;\; (m{\leq}n):$$

$$\boldsymbol{k} \boxplus (\boldsymbol{x} \overset{def}{=} s_0 s_1 \cdots s_{n-1}) = \begin{bmatrix} s_{n-1} & \cdots & s_{k-1} & \cdots & s_1 & s_0 \\ s_n & \ddots & \cdots & \ddots & s_2 & s_1 \\ \vdots & \ddots & \ddots & \cdots & \ddots & \ddots \\ s_{n+m-2} & \cdots & s_n & s_{n-1} & \cdots & s_{m-1} \end{bmatrix} \cdot \boldsymbol{k}.$$

Hence, the function "$\boxplus$" is equivalent to:

$$\boxplus : \boldsymbol{k} \times \boldsymbol{x} \mapsto [\langle \boldsymbol{x} \cdot A^0, \boldsymbol{k} \rangle, \langle \boldsymbol{x} \cdot A^1, \boldsymbol{k} \rangle, \cdots, \langle \boldsymbol{x} \cdot A^{m-1}, \boldsymbol{k} \rangle], \tag{2}$$

where the matrix A is defined as follows:

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & a_0 \\ 1 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 & a_{n-2} \\ 0 & 0 & \cdots & 1 & a_{n-1} \end{bmatrix}. \tag{3}$$

This function is a 2-source extractor since the Toeplitz matrix (as in the definition of $\boxplus$) has full rank for any non-zero vector $\boldsymbol{x}$, which in turn follows from the properties of maximal length LFSR. Note, that this extractor directly inherits the homomorphic property of Krawczyk's hash function. Namely, we have that $\langle \boldsymbol{x} \cdot A^i, \boldsymbol{k} + \boldsymbol{m} \rangle + \langle \boldsymbol{x} \cdot A^i, \boldsymbol{m} \rangle = \langle \boldsymbol{x} \cdot A^i, \boldsymbol{k} \rangle$.

## 3 Hardware implementation

Following the specification from the previous section, we now present the hardware architecture and the tradeoffs that we considered when implementing the Hadamard extractor. We also use this description of the hardware to list the different parameters that will be analyzed in our following side-channel evaluations. As indicated by the notations in Section 2, we will generally apply the extractor to an $n$-bit public value $\boldsymbol{x}$ and an $n$-bit secret key $\boldsymbol{k}$, in order to produce an $l$-bit random string $y$ (which is the typical scenario in leakage resilient cryptography). Practical values that we consider in this work are $n = 192$ and $l = 128$. For simplicity, we start by describing a fully serial implementation (with $n = 8$), illustrated in Figure 1.

In this basic form, the extractor circuit mainly consists of two registers. One is used to store the current LFSR values (denoted as $r[0]$ to $r[7]$ in the figure), and consequently evolves as the implementation is running. The other one is used to store the key and remains static during the extraction process. Note that the decision to store the secret key in the static register is motivated by the minimization of the computations (hence, leakage) involving secret data. In a

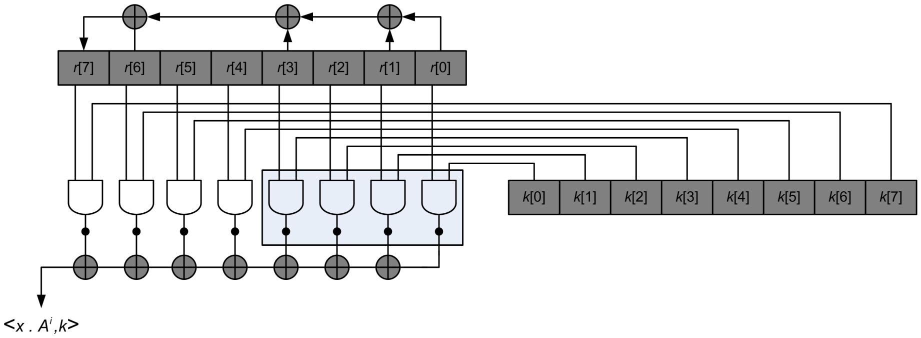

**Fig. 1.** Fully serial hardware implementation of the extractor.

fully serial implementation, the $r$ register is shifted by one position at each clock cycle. Next, $n$ AND gates and $n-1$ XOR gates are added to the design, in such a way that one can extract one bit (i.e. compute one inner product $\langle \boldsymbol{x} \cdot A^i, \boldsymbol{k} \rangle$) per clock cycle. Thus, in order to extract 128 bits, we have to clock the circuit 128 times (while the registers are typically 192-bit long). Such a basic design can essentially be extended in two main directions that we now detail.

**Parallelizing the implementation.** In general, hardware implementations are most efficient if they can take advantage of some inherent parallelism in algorithms. Fortunately, this is typically the case when considering our extractor. That is, as illustrated in Figure 2, one can easily double the throughput of the previous design, by extending the LFSR by one cell $r[8]$ and duplicating the combinatorial parts of the design (i.e. the XOR gates used in the LFSR recurrence and the inner product computation). This allows one to compute two inner products per clock cycle. Interestingly, the registers cells $r[1]$ to $r[7]$ and the key register can be shared by these two inner product combinations, which makes the parallelization quite efficient. Such a process can be further extended. In general, by multiplying the number of inner product combinations $p$ times, we decrease the number of cycles to extract 128 bits by the same factor.

**Masking the implementation.** Next, as detailed in Section 2, the proposed extractor inherently benefits from an additive homomorphic property. This implies that it can be easily masked, following the proposals of Goubin and Patarin [7] and Chari et al. [1]. In our setting, it is most natural to mask the key, as masking the plaintext would lead to a weakness similar to the "zero problem" when applying multiplicative masking to the AES S-box [6]. That is, the bitwise AND between a masked plaintext and a key would still allow distinguishing the zero key bits. From an implementation point of view, a masked computation $\langle \boldsymbol{x} \cdot A^i, \boldsymbol{k} + \boldsymbol{m} \rangle + \langle \boldsymbol{x} \cdot A^i, \boldsymbol{m} \rangle = \langle \boldsymbol{x} \cdot A^i, \boldsymbol{k} \rangle$ can be performed using essentially the same design as in the unmasked case. And it straightforwardly extends to

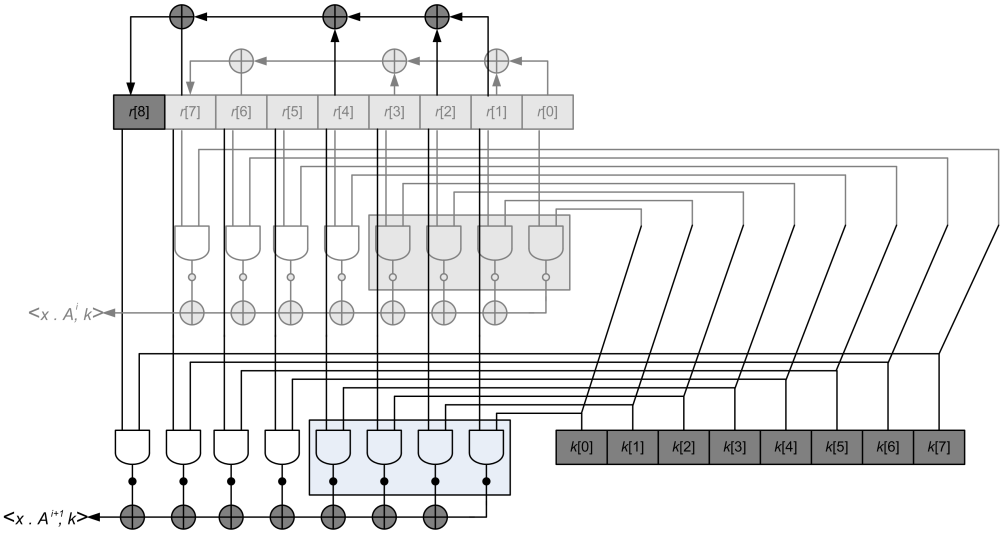

**Fig. 2.** Hardware implementation of the extractor with parallelism ($p = 2$).

higher-order masking, where the order $o$ of the masking scheme refers to the number of $n$-bit masks consumed per extraction, as we now detail.

First observe that, in the unprotected case, the result of an extraction is available after $128/p$ clock cycles. In the masked case, this performance decreases only linearly, since it is possible to operate on all shares independently. In other words, each mask can be discarded immediately after it has been processed. For this purpose, we first need 192 clock cycles to load the mask register and (at the same time) add the mask to the key. Next, 128 clock cycles are needed to extract from the mask. Finally, the (bidirectional) plaintext register is rewound during 128 clock cycles. Overall, every mask needs 448 clock cycles to be processed. And the final result is obtained by extracting from the masked key, which requires another 128 clock cycles. Summarizing, we have implemented the circuit such that the cycle count $c$ increases linearly, following the formula $c = (128+o\cdot 448)/p$, with $o$ the masking order and $p$ the degree of parallelization. Table 1 summarizes the performances of various extractor implementations. These numbers were obtained from post-synthesis results, using Cadence RTL compiler 2009 and the UMC F180GII standard-cell library. Note that, in a fully serial and unprotected implementation, the area cost is already dominated by the registers $r$ and $k$. They alone account for 3.6 kGE. Roughly speaking, the hardware overhead of a masked implementation mainly corresponds to one additional register for storing the mask, and a 192-bit multiplexer in order to switch the AND gates' input between the key and the mask.

**Table 1.** Area in kilo gate equivalents (kGE) and cycle count (c) for extractor implementations with different datapath widths and different masking orders (the polynomial $x^{192} + x^{149} + x^{97} + x^{46} + 1$ has been used in the recurrence).

| Parallelization | 1 | 4 | 8 |
|---|---|---|---|
| w/o masking | 4.3 kGE 128 c | 7.0 kGE 32 c | 10.3 kGE 16 c |
| $1^{st}$-order | 7.3 kGE 576 c | 10.1 kGE 144 c | 13.6 kGE 72 c |
| $2^{nd}$-order | 7.3 kGE 1024 c | 10.1 kGE 256 c | 13.6 kGE 128 c |
| $3^{rd}$-order | 7.3 kGE 1472 c | 10.1 kGE 368 c | 13.6 kGE 184 c |

## 4 Adversarial capabilities and leakage assumptions

The goal of this paper is to investigate the side-channel resistance of different versions of the implemented extractor, with and without parallelism and masking. For this purpose, we will apply the two parts of the framework in [15]. That is, we start with an information theoretic analysis (in the next section), in order to capture a worst case scenario. Next, we perform a security analysis, that considers the success rates of different adversaries. In general, such an evaluation requires to define the adversary's capabilities and leakage assumptions.

In our present context, the first question to answer is to determine the target operations for the side-channel adversary. For this purpose, one generally selects the operations where the known input $\boldsymbol{x}$ and secret key $\boldsymbol{k}$ are mingled. For the extractor implementations in Figures 1 and 2, this corresponds to the bitwise AND gates (the side-channel attacks against a software implementation of extractor in [14] were based on exactly the same assumption). Next, it is typically needed to determine the size of the key guess (i.e. the number of key candidates that will be enumerated in the attack). In the following, we will consider a 4-bit key guess, that is a convenient choice for limiting the time complexity of the evaluations. This choice is motivated by the fact that we aim to investigate many sets of parameters. In the Figures 1 and 2, it means that an adversary will typically try to predict the output of the AND gates that are included in the gray rectangles.

In addition, and more importantly, a central feature of the Hadamard extractor implementations is that the key register is used numerous times in order to extract $l$ bits. For example, in the serial implementation of Figure 1, in which one extracts $l = 128$ bits, it implies that 128 leaking operations can potentially be exploited by the adversary. In the case of the parallel implementation of Figure 2, where $p = 2$, this amount of exploitable leakage points is decreased to 64. This is in strong contrast with traditional implementations of block ciphers, where one typically predicts a few leakage points, corresponding to the intermediate computations of the block cipher that can be easily enumerated. For example, in the block cipher PRESENT, a 4-bit guess allows predicting the first (or last) key addition and S-box computations of an encryption process. But following operations become hard to predict, because of the diffusion in the cipher. As implementations of extractors are not affected by such a strong diffusion property, a very important parameter is the number of leakage points exploited by the adversary. Our experiments will consider both single-sample attacks, that are similar to standard DPA attacks against block ciphers, and $t$-sample attacks, for which we aim at discussing their relevance in a side-channel evaluation context. Summarizing, our evaluations will investigate three main parameters:

1. the degree of parallelism in the implementation $p$,
2. the order of the masking scheme $o$,
3. the number of leakage samples per plaintext exploited in the attacks $t$.

Our experiments are based on simulated traces, which reflect the ideal power consumption of the previously described hardware architecture. In order to allow a systematic comparison between the level of security of the extractor implementation and the one of a masked S-box, we added Gaussian noise. More precisely, we used simulated traces to generate the mean value of the target leakages, with the Signal-to-Noise Ratio as a parameter ($\mathrm{SNR} = 10 \cdot \log_{10}(\frac{\sigma_s^2}{\sigma_n^2})$). In other words, we extended the simulation environment of [17] to the context of an extractor. Note, that most of our following conclusions relate to the comparative impact of the parameters $p, o$ and $m$. Hence, the possible deviations that one would observe between simulated traces and actual measurements would not affect these conclusions (i.e. they would essentially only cause some slight shifts of the information theoretic and security analysis curves in the following sections).

## 5 Information theoretic analysis

In this section, we aim to evaluate the security of the previously described extractor implementation, in function of the amount of parallelism, masking and leakage samples available to the adversary. For this purpose, we start with the information theoretic analysis advocated in [15], the goal of which is to analyze a worst case scenario, where the adversary has perfect knowledge of the leakage distribution (i.e. is able to perform a perfect profiling). For our simulated setting, this means that the adversary is provided with the leakage samples $l_j^i$, where the subscript $j$ relates to the number of shares in a masking scheme and the superscript $i$ relates to the number of samples used per input $x$ in the attack. More specifically, in an unmasked implementation, the adversary is given the following leakage samples:

$$l_1^i = \mathrm{W_H}\left((\boldsymbol{x} \cdot A^i) \wedge \boldsymbol{k}\right) + n, \tag{4}$$

where $\mathrm{W_H}$ denotes the Hamming weight function and $n$ is a Gaussian noise. From this definition, one can straightforwardly compute the following mutual information metric, for the fully serial (i.e. $p = 1$) single sample (i.e. $t = 1$) case:

$$\mathrm{I}(K; X, L_1^1) = \mathrm{H}[K] - \sum_{k \in \mathcal{K}} \Pr[k] \sum_{x \in \mathcal{X}} \Pr[x] \sum_{l_1^1 \in \mathcal{L}} \Pr[l_1^1 | x, k] \cdot \log_2 \Pr[k | x, l_1^1]. \tag{5}$$

Next, considering a masked implementation would change the previous analysis as follows. First, the adversary now has to exploit the leakage of several shares. For example, in the first-order case (i.e. $o = 1$), we have:

$$l_1^i = \mathrm{W_H}\left((\boldsymbol{x} \cdot A^i) \wedge \boldsymbol{m}\right) + n, \tag{6}$$
$$l_2^i = \mathrm{W_H}\left((\boldsymbol{x} \cdot A^i) \wedge (\boldsymbol{k} \oplus \boldsymbol{m})\right) + n. \tag{7}$$

Second, the computation of the mutual information metric is turned into:

$$\begin{aligned}\mathrm{I}(K; X, L_1^1, L_2^1) = \mathrm{H}[K] - \sum_{k\in\mathcal{K}} \Pr[k] \sum_{x\in\mathcal{X}} \Pr[x] \sum_{m\in\mathcal{M}} \Pr[m]& \\ \cdot \sum_{l_1^1, l_2^1 \in \mathcal{L}^2} \Pr[l_1^1, l_2^1 | x, m, k] \cdot \log_2 \Pr[k | x, l_1^1, l_2^1],&\end{aligned} \tag{8}$$

where $\Pr[k|x, l_1^1, l_2^1] = \sum_{m'} \Pr[m'|x, l_1^1] \Pr[k|x, m', l_2^1]$. That is, the mask is not given to the adversary, but its leakage allows building a bivariate conditional distribution that is key-dependent. This naturally extends towards larger $o$'s.

These previous equations were considering single-sample-per-input attacks that are typically similar to the DPA against the AES S-box in [10] and masked AES S-box in [17]. When moving to the multi-sample context, the computation of the mutual information metric for the unmasked case is turned into:

$$\begin{aligned}&\mathrm{I}\left(K; X, L_1^t\right) = \mathrm{H}[K] - \sum_{k\in\mathcal{K}} \Pr[k] \sum_{x\in\mathcal{X}} \Pr[x] \\ &\cdot \sum_{l_1^1, l_1^2, \ldots, l_1^t \in \mathcal{L}^t} \Pr[l_1^1, l_1^2, \ldots, l_1^t | x, k] \cdot \log_2 \Pr[k | x, l_1^1, l_1^2, \ldots, l_1^t].\end{aligned} \tag{9}$$

And for the masked case, it becomes:

$$\begin{aligned}&\mathrm{I}\left(K; X, L_1^t, L_2^t\right) = \mathrm{H}[K] - \sum_{k\in\mathcal{K}} \Pr[k] \sum_{x\in\mathcal{X}} \Pr[x] \sum_{m\in\mathcal{M}} \Pr[m] \\ &\cdot \sum_{l_1^1, l_1^2, \ldots, l_1^t, l_2^1, l_2^2, \ldots, l_2^t \in \mathcal{L}^{2t}} \Pr[l_1^1, l_1^2, \ldots, l_1^t, l_2^1, l_2^2, \ldots, l_2^t | x, m, k] \\ &\cdot \log_2 \Pr[k | x, l_1^1, l_1^2, \ldots, l_1^t, l_2^1, l_2^2, \ldots, l_2^t].\end{aligned} \tag{10}$$

Interestingly, this multi-sample case implies that many samples can be used to "bias" the mask in the computation of the mixture distribution:

$$\Pr[k|x, l_1^1, l_1^2, \ldots, l_1^t, l_2^1, l_2^2, \ldots, l_2^t] = \sum_m \Pr[m|x, l_1^1, l_1^2, \ldots, l_1^t] \Pr[k|x, m, l_2^1, l_2^2, \ldots, l_2^t].$$

As will be seen in the following, this strongly reduces the security improvements of masking in this setting. Finally, independent of the parameters $o$ and $t$, an increased parallelism is modeled by changing the leakage function. For example, in the $p = 2$ case of Figure 2, the adversary would obtain samples of the form:

$$l_1^{i,i+1} = \mathrm{W_H}\left((\boldsymbol{x} \cdot A^i) \wedge \boldsymbol{k}\right) + \mathrm{W_H}\left((\boldsymbol{x} \cdot A^{i+1}) \wedge \boldsymbol{k}\right) + n. \tag{11}$$

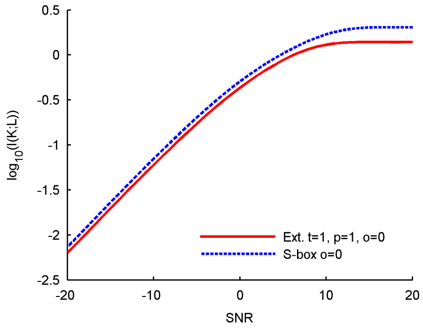

**Fig. 3.** Single sample attacks, serial implementation, without masking.

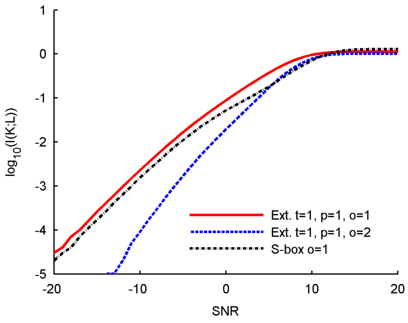

**Fig. 4.** Single sample attacks, serial implementation, with masking.

In general, modifying the parallelism has no impact on the previous equations. However, increasing $p$ implies that the maximum number of samples that one can exploit per plaintext is more limited (to 64 if $p = 2$, 32 if $p = 4, \ldots$). Note again that, due to the weak diffusion of the extractor implementation, a 4-bit guess would then allow to predict several 4-bit parts of the inner product computations (e.g. one part of both $\mathrm{W_H}$ functions in Equation (11) can be predicted).

### 5.1 Single sample attacks, serial implementation

For the first scenario, we assume an adversary who looks only at one leakage sample per side-channel trace. This can be seen as a naive attack, where the adversary applies exactly the same strategy as he would for an S-box. The implementation of the extractor is fully serialized and its masking order varies between zero and two. Figure 3 shows the unprotected case ($t = 1$, $p = 1$, $o = 0$) and Figure 4 shows the masked case ($t = 1$, $p = 1$, $o \in \{1, 2\}$). We compare those curves with the information curves for an unmasked and a masked PRESENT S-box. As higher-order masking schemes leading to efficient hardware implementations remain an open problem, we restrict the S-box evaluations to first-order masking[1]. For the unmasked case, the curves confirm what was already observed in [14]. Namely, a single extractor sample contains less information than a single S-box sample, on average. As for the masked case, the results follow the expectations in [1, 17]. That is, for a sufficient amount of noise, increasing the order of the masking scheme implies an exponential security increase, reflected by the different slopes of the log scale curves in Figure 4. Note finally that, in this latter case and for similar orders, the information provided by a single sample of the extractor is now slightly higher than the one provided by an S-box sample (which can be explained by the shape of the masked leakage distributions).

[1] The only straightforward solution is to use a large look-up table of size $2^{o \cdot n} \times n$ bits.

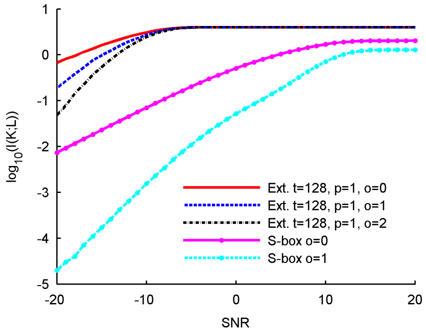

**Fig. 5.** Multi sample attacks, serial implementation.

### 5.2 Multi-sample attacks, serial implementation

The results in the previous section suggest that the security of an extractor implementation can be strong when adversaries exploit a single sample per leakage trace. But as previously mentioned, extractors are different than standard block ciphers in the sense that a small key guess (here 4-bit) allows adversaries to predict multiple intermediate computation results (up to $t = 128$). In this section, we consider the worst case of a serial implementation where an adversary would exploit all this information. Applying this approach to the unmasked case implies to compute the mutual information metric given in Equation (9). We mention that, since integrating over 128 dimensions is too complex, the following estimations are obtained by statistical sampling. Similarly, evaluating a multi-sample attack against a masked implementation requires to compute Equation (10). As previously mentioned, this equation suggests that the mask can be strongly biased because, in the multi-sample attack setting, an adversary can exploit 128 leakage points generated from the manipulation of the same mask value.

The results of the information theoretic analysis corresponding to this strongest possible adversary are depicted in Figure 5. It can be seen that the situation changes dramatically. Up to an SNR of -5, the remaining entropy of the key variable after seeing a single side-channel trace is zero (i.e. unbound leakage). For SNRs below -5 the recovered information eventually decreases and also masking starts to bring additional security. However, due to the mask biasing process, it also requires smaller SNRs until the impact of masking is fully released. Furthermore, even for an SNR as small as -20, the second-order masked extractor implementation reveals more information than the unprotected S-box one. Summarizing, while an extractor implementation provides strong security against standard univariate DPA attacks, the exploitation of multiple samples leads to an opposite conclusion. Roughly speaking, the exploitation of $t$ samples per trace in this setting corresponds to the exploitation of $t$ single-sample traces (all using the same mask) in the previous section. We now discuss strategies to relax this limitation.

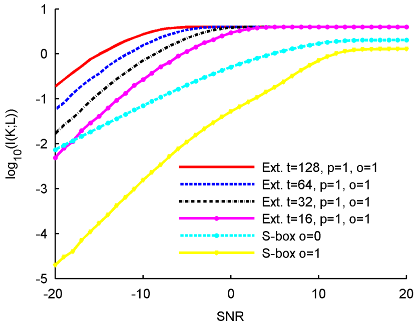

**Fig. 6.** Multi-sample attacks, reducing $t$ for masking of order 1.

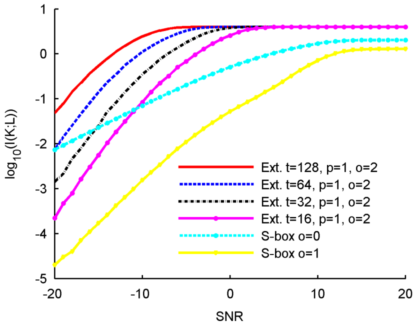

**Fig. 7.** Multi-sample attacks, reducing $t$ for masking of order 2.

### 5.3 Decreasing the leakage by reducing $t$

Following the previous section, one important objective for improving the security of an extractor implementation is to limit the number of leakage points exploitable per trace. In the full version of this paper [11], we investigate different strategies to achieve this goal, namely re-keying, re-masking and parallelism and compare them from an implementation and side-channel point of view. Here, we focus on the general effect of reducing the parameter $t$ and in particular discuss parallelization as it is the most appealing approach from a performance point of view.

Figures 6 and 7 show the impact of limiting $t$ in such a way, for masking of order one and two. It can be seen that the information decreases exponentially with $t$. Furthermore, this exponential decrease is larger the higher the masking order is, essentially because we do not only limit the available samples for the key but also for the masks.

In general, parallelization has the same impact as just reducing $t$. However, as now more computations are performed per clock cycle, also the amount of information leakage contained per sample might be higher. As a simple example, let us denote two leakage samples generated by an unprotected serial implementation as $l_1$ and $l_2$. By parallelizing the operations corresponding to these leakage samples, one provides the adversary with a new sample $l' = l_1 + l_2$. If these two leakage samples are related to the same key guess, the information provided by one of them is generally less than the information provided by their sum, which is again less than their joint information. This is illustrated in Figures 8 and 9. However, as will be seen in the security analysis, not every distinguisher is capable of exploiting this extra information.

## 6 Security analysis

The results of the previous IT analysis define upper bounds for the information which can be extracted by an adversary. In this section we discuss how well these upper bounds represent the capabilities of an adversary. In particular, we raise two questions: (1) Are $t$-sample attacks relevant in practice? (2) How large is the practical security gain of masking an extractor? The first question can be answered positively under some reasonable assumptions. As for the second question, it turns out that masked extractors can actually provide good security when looking at standard higher-order DPA attacks.

### 6.1 Identifying multiple samples

In general, the critical parameters for evaluating side-channel attacks are the data complexity (in the first place) and the time complexity (when the order of the attacks increases). The data complexity typically depends on the amount of information contained in each leakage trace. The time complexity typically depends on the number of samples of interest to identify in the traces. As a result, it is interesting to determine how efficiently the previous multi-sample attacks trade time and data. For this purpose, say an adversary has to identify $t$ samples in an $N$-sample leakage trace. Without additional assumptions, the complexity of finding them is in $O(N^t)$. However, in the case of the hardware extractor, the adversary can assume that time samples are equidistantly distributed along the power trace. For instance, our hardware architecture produces $p$ bits every clock cycle and the extraction process takes $t$ clock cycles. Thus the distance between the interesting samples is one clock cycle. Given this information, one can sweep over the power trace like in a standard single-sample DPA, and directly launch a multi-sample attack exploiting the $t$ equidistant samples with a complexity in $O(N)$. In other words, an adversary does not have to detect these samples of interest separately and in advance. This observation implies the interesting consequence that the order of a side-channel attack (usually defined as the number of samples exploited per trace) is not a generally good indicator of security. It is sometimes easy to launch high-order attacks. Note also that the assumptions on the underlying hardware to launch such low complexity attacks are generally

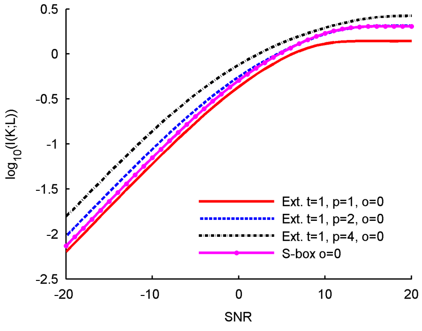

**Fig. 8.** Single sample attacks with parallelism, no masking.

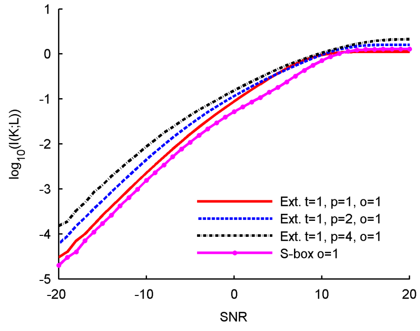

**Fig. 9.** Single sample attacks with parallelism, masking order 1.

easy to guess for adversaries. Hence, ruling out multi-sample attacks in this case implies to rely on "security by obscurity" in an excessive manner.

### 6.2 Attacking the masking

An interesting feature of our extractor implementation is that its homomorphic property allows efficient masking of unusually large orders (compared, e.g. to the AES Rijndael). However, as previously discussed, the impact of masks can be strongly reduced in multi-sample attacks, as an adversary can theoretically bias the mask using all the available samples. In this section, we discuss the feasibility of such a biasing in a non-profiled attack setting.

**Biasing the mask.** As detailed in Section 5.2, biasing the masks is straightforward in profiled side-channel attacks. The adversary just has to launch a template attack on the masks, prior to the attack, and can use the resulting distribution of the masks when launching the same template attack on the secret key. By contrast, when moving to a non-profiled attack setting, such a mask biasing becomes more difficult to exploit. The natural approach would again be to launch a DPA (e.g. correlation-based) on the mask extraction $\langle \boldsymbol{x} \cdot A^i, \boldsymbol{m} \rangle$. If the measurements are quite informative, then this DPA can lead to a complete recovery of the masks (i.e. the masked implementation becomes as easy to break as the unprotected one). But when measurements become noisy, the adversary ends up with a vector of key candidates ranked according to the output of the DPA distinguisher, e.g. a correlation coefficient. Exploiting this information in a non-profiled attack has to be based on one of the following approaches:

1. If only a couple of mask candidates remain likely after the biasing process, one solution is to test them exhaustively. But this strategy becomes intensive when combining multiple plaintexts, as the number of such tests scales exponentially in the number of plaintexts in the attack.
2. If the mask distribution obtained after the observation of a trace is not enough biased, or if the number of plaintexts required to perform a successful key recovery is too high, one then has to rely on heuristics. For example, one solution would be to normalize the correlation coefficient obtained after the DPA against the masks, and to interpret them as probabilities. As detailed in [18], this heuristic already implies significant efficiency losses compared to the application of a template attack. In addition, one then needs to exploit this vector of mask "probabilities" in the high-order attack against the extractor implementation, which is again not trivial. For example, one could try to exploit this information to improve the combination function used in a second-order DPA, as suggested in [13]. But it usually results in quite involved techniques that may not be easy to apply in practical settings.

As in the previous subsection, it is interesting to observe that the evaluation of the extractor implementation crucially relies on the strategy adopted by the adversary. But while the multi-sample approach is quite realistic in a hardware

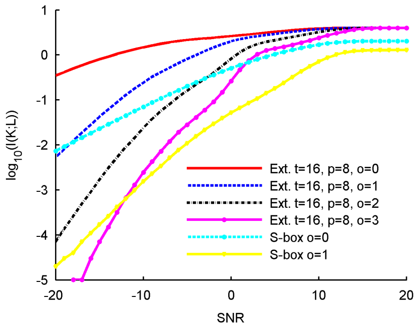

**Fig. 10.** Information leakage without mask biasing (p=8, t=16).

implementation context, even for non-profiled attacks, the biasing of the masks becomes much less realistic when profiling the chip and taking advantage of Bayesian key recovery is not possible. In fact, a more practical adversary will probably perform a higher-order DPA directly on the different time samples provided by the traces (for instance by performing a Pearson-correlation based attack after applying the normalized product combining function [9] or by directly using a multivariate distinguisher like MIA [5]). Interestingly, one can easily evaluate the information loss caused by this strategy, by replacing the last factor in Equation (10) by:

$$\Pr[k|x, l_1^1, l_1^2, \ldots, l_1^t, l_2^1, l_2^2, \ldots, l_2^t] = \frac{\prod_{i=1}^{t} \sum_m \Pr[m|x, l_1^i] \Pr[k|x, m, l_2^i]}{\sum_{k'} \prod_{i=1}^{t} \sum_m \Pr[m|x, l_1^i] \Pr[k'|x, m, l_2^i]}.$$

The result of such an evaluation for $t = 16$ and a parallelization of 8 is plotted in Figure 10. Intuitively, it represents the upper bound for the information which can be retrieved by a non-biased DPA attack. In a typical scenario where an adversary attacks 8 out of 128 bits at once, the SNR can be computed as $10 \log_{10}(8/120) = -11.761$. It can be seen from the figure that this is approximately the point where a profiled adversary does not gain more information from a multi-sample attack on the $3^{rd}$-order masked extractor than from a single-sample attack on a $1^{st}$-order masked S-box.

**Higher-order DPA attacks.** Eventually, it is interesting to evaluate how efficiently a standard higher-order DPA against the masked extractor implementation can take advantage of the information leakage in Figure 10. For this purpose, and based on the fact that our implementation exhibits a perfect Hamming weight leakage model, it is natural to apply a DPA based on Pearson's correlation coefficient, and using a normalized-product combining function. In Figure 11, we compare a $1^{st}$-order masked S-box and a $3^{rd}$-order masked extractor for SNRs of 0 and 10. One can observe that the results of this security evaluation do not directly reflect the outcome of the IT analysis. Namely, even

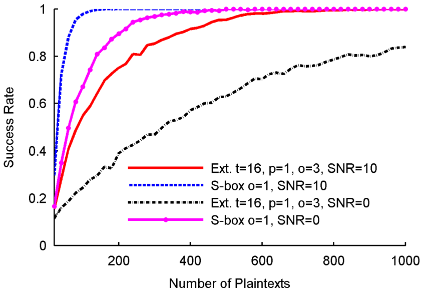

**Fig. 11.** Multi-sample correlation attacks against masked implementations, normalized product combining.

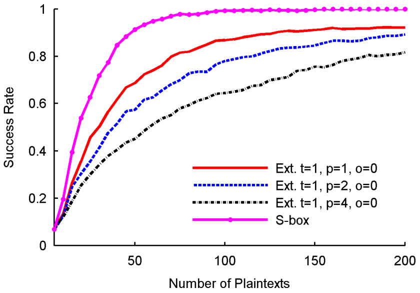

**Fig. 12.** Single-sample correlation attacks against unprotected implementations, with an SNR=-0.3.

in the low noise scenario, the multi-sample attack on the extractor is almost a magnitude less efficient than the single-sample attack on the S-box, due to the information loss caused by the normalized-product combining. Interestingly, even in an unprotected context, the sub-optimality of DPA attacks can be highlighted, e.g. in Figure 12, where parallelism always improves security[2].

We finally note that, more than the results of specific adversarial strategies used in this section (which may be suboptimal), it is the very observation that these strategies play a central role in the security evaluation of cryptographic devices that is interesting. Especially, if the leakage of the analyzed primitive has flavors unknown from block ciphers, such as multiple samples per plaintext and inherent mask re-use, special care has to be taken during the evaluation and the interpretation of the results.

## 7 Conclusions

This paper first shows the interest of implementing randomness extractors in hardware. By taking advantage of different design tradeoffs (namely, parallelism, re-keying and re-masking), such implementations allow improving the security against side-channel attacks, in particular when compared to their software counterpart. Next, our results put forward the strong impact of adversarial strategies in the evaluation of extractor implementations (and leaking devices in general). Depending on the capabilities of an adversary, the information leakage provided by the extractor implementations considered in this paper range from large (in Figure 5) to much more limited (in Figure 10) and is sometimes difficult to exploit with standard DPA attacks (in Figure 11). From a methodological point of view, this observation suggests to always consider different capabilities in an evaluation, in order to avoid overestimating (or underestimating) the security of an implementation. Our results also show that the order of an attack, defined

[2] In contrast with the result given in Section 5.3, Figure 8.

as the number of samples per trace, may not be a good indication of security if these samples do not corresponds to the different shares of a masking scheme. As a scope for further research, we notice that the main weakness of the extractor investigated in this paper derives from the ability to predict many intermediate computations from a small key guess, which reduces the interest in its nice homomorphic property. As a result, it would be interesting to design an extractor limiting this weakness, e.g. by introducing a type of "key-scheduling" in the algorithm, in order to add some diffusion during the extraction process.

## References

1. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
2. Y. Dodis and K. Pietrzak. Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2010.
3. S. Dziembowski and K. Pietrzak. Leakage-Resilient Cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
4. S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In D. Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010.
5. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
6. J. D. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.
7. L. Goubin and J. Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and C. Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
8. H. Krawczyk. LFSR-based Hashing and Authentication. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.
9. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer, 2007.
10. S. Mangard, E. Oswald, and F.-X. Standaert. One for All - All for One: Unifying Standard DPA Attacks. Cryptology ePrint Archive, Report 2009/449, 2009. `http://eprint.iacr.org/`, to appear in IET Information Security.
11. M. Medwed and F.-X. Standaert. Extractors Against Side-Channel Attacks: Weak or Strong? Cryptology ePrint Archive, Report 2011/348, 2011. `http://eprint.iacr.org/`.
12. M. Naor and G. Segev. Public-Key Cryptosystems Resilient to Key Leakage. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.
13. E. Prouff, M. Rivain, and R. Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.

14. F.-X. Standaert. How Leaky Is an Extractor? In M. Abdalla and P. S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 294–304. Springer, 2010.
15. F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
16. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, and E. Oswald. Leakage Resilient Cryptography in Practice. In D. Basin, U. Maurer, A.-R. Sadeghi, and D. Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer Berlin Heidelberg, 2010.
17. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.
18. N. Veyrat-Charvillon and F.-X. Standaert. Adaptive Chosen-Message Side-Channel Attacks. In J. Zhou and M. Yung, editors, *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 186–199, 2010.
19. Y. Yu, F.-X. Standaert, O. Pereira, and M. Yung. Practical leakage-resilient pseudorandom generators. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 141–151. ACM, 2010.