How Leaky is an Extractor?

François-Xavier Standaert[⋆]

Université catholique de Louvain, Crypto Group, Belgium.
e-mails: `fstandae@uclouvain.be`

Abstract. This paper discusses the security of a leakage-resilient stream cipher presented at FOCS 2008, instantiated in a practical setting. Based on a case study, we put forward implementation weaknesses that can be exploited in a key-recovery attack. We first show that in our experimental context (8-bit device, Hamming weight leakages, Gaussian noise), a successful attack against the investigated stream cipher has lower data complexity than a similar attack against an unprotected AES implementation. We then analyze the origin of the observed weaknesses and relate them with the implementation of extractor that is used in the investigated stream cipher. We finally discuss the implications of these results for the design of leakage-resilient primitives and provide guidelines to improve the construction of FOCS 2008 and its underlying components.

1 Introduction

In a side-channel attack, an adversary attempts to break a cryptographic primitive, by taking advantage of the physical peculiarities of the hardware on which it is running. Typical examples include the power consumption or electromagnetic radiation of small embedded devices. In view of the physical nature of these implementation issues, the first countermeasures to prevent them were mainly designed at the hardware level. But recent results have witnessed a growing interest of the cryptographic community to extend the applicability of provably secure constructions in this new setting. For example, two constructions of leakage-resilient stream ciphers have recently been proposed, at FOCS 2008 and Eurocrypt 2009. The first one is based on the combination of a pseudorandom generator (PRG) and an extractor [4], while the second one only uses a block cipher based PRG as building block. Both constructions are proven leakage-resilient in the standard model, the extractor based construction allowing significantly better security bounds. Both constructions rely on the fact that the amount of information leakage in one iteration of the cipher is bounded in some sense - which has to be guaranteed by hardware designers. This bound is usually referred to as a λ-bit leakage, intuitively meaning that only a part of the internal parameters in the target device is revealed by the physics. Different metrics can be used to quantify this bounded information, *e.g.* [4, 7] use the HILL pseudoentropy. In view of the strong nature of these security claims (in particular

[⋆] Research Associate of the Belgian Fund for Scientific Research (FNRS - F.R.S.).

compared to the limited results provided by hardware-level countermeasures), it is natural to challenge these theoretical constructions, by investigating and quantifying their security properties in actual leaking devices.

In this paper, we consequently study the practical security of the FOCS 2008 stream cipher in an 8-bit device, against standard DPA. In particular, and as a usual scenario for evaluating countermeasures against side-channel attacks, we consider a Hamming weight leakage model with Gaussian noise. We use this case study to discuss some issues left unanswered by theoretical analysis, namely:

– The construction in [4] is based on an extractor, but the actual instance of extractor to use in a practical implementation is left unspecified. Hence, the good selection of an extractor in this context is an open question.

– Given that an extractor is specified, does the addition of this new primitive in the hardware have an impact on the λ-bit leakage assumed in the proofs? And how does this different λ modify the global security of the construction?

Our simulated experiments clearly indicate that if no particular attention is paid, the extractor may be the weak point in this stream cipher construction. More precisely, the success rate of a standard DPA against *one* iteration of the extractor in our target device is (much) higher than the one of a similar standard DPA attack against *several* iterations of an unprotected AES implementation. This is due to the fact that extracting several random bits from weak sources, as required in [4], implies to manipulate the same secret data several times. In other words, the implementation of the extractor has a strong impact on the λ-bit leakage and eventually gives rise to little actual security for the PRG in this case. We then discuss the consequences of these results and provide guidelines, both for improving the selection of the extractors to use in leakage-resilient cryptography and for improving the PRG construction of FOCS 2008.

We note that our results are not in contradiction with [4, 7] but emphasize the gap existing between the assumptions of these theoretical works and the specificities of actual implementations. That is, any single gate added in a hardware device, or single line added in a software code, have a potential impact on the information leakages. Hence, it is difficult (if not impossible) to discuss the physical security of a primitive without a precise understanding of the relations between its algorithmic description and its implementation properties. This is especially true when considering primitives such as extractors for which very few experimental attacks have been performed, *e.g.* compared to block ciphers such as the AES Rijndael. That is, as acknowledged by the authors of [4, 7], providing means to exploit a bounded leakage per iteration in a stream cipher construction solves only one half of the side-channel issue. It then remains to ensure that a good leakage bound can be guaranteed in practice, using existing hardware. In this paper, we additionally show that the good understanding and evaluation of the low-level characteristics in physical leakages may also motivate better choices for the selection of algorithms to use in order to best face side-channel attacks.

The rest of this paper is structured as follows. Some background information is given in Section 2. The implementation of the FOCS 2008 stream cipher that we target is described in Section 3. Our security analysis and experimental results are in Section 4. Eventually, Section 5 provides guidelines to improve the design and analysis of leakage-resilient primitives and Section 6 concludes the paper.

2 Background

The construction. Our analysis focuses on an instantiation of the leakage-resilient stream cipher of FOCS 2008, represented in Figure 1, in which ext denotes a two-source extractor and PRG a length-tripling Pseudo-Random number Generator[1]. The components of this construction are as follows.

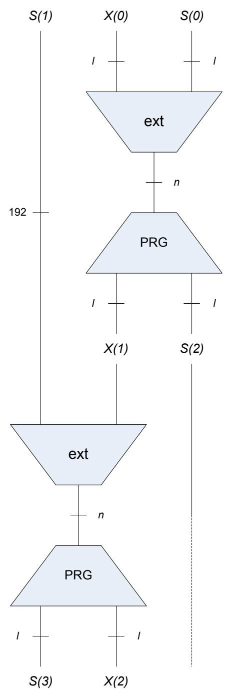

Fig. 1. Leakage-resilient stream cipher from FOCS 2008.

[1] This construction does not directly correspond to the one of Dziembowski and Pietrzak. In [4], Figure 2, the extractor's output is split in two parts: $(K_{i+1}, X_{i+1}) = \mathsf{ext}(K_i, B_i)$, and only one part is sent to the PRG: $B_{i+1} = \mathsf{PRG}'(X_{i+1})$. In our description, the extractor's output is fully transmitted to the PRG. But the instance in Figure 1 can be seen as a generalization of the FOCS stream cipher by defining our PRG as $\mathsf{PRG}(K_{i+1}, X_{i+1}) = K_{i+1}||\mathsf{PRG}'(X_{i+1})$, using notations from [4], which also makes an easy connection to the Eurocrypt 2009 proposal.

Length-tripling PRG. For convenience, and because they are usual targets in side-channel attacks, we use a length tripling PRG that is based on a block cipher. Let $\mathsf{AES}_k(x)$ denote the encryption of a plaintext x under a key k, we have:

$$\mathsf{PRG} : \{0,1\}^n \mapsto \{0,1\}^{3n} : x \mapsto \Big(\mathsf{AES}_x(c_1), \mathsf{AES}_x(c_2), \mathsf{AES}_x(c_3)\Big),$$

where c_1, c_2, c_3 are three public constant values.

Two-source extractor. We consider the constructions in [3]. They allow extracting many random bits from two weak sources X, Y of the same length l. Let $A_1, A_2, \ldots, A_n$ be $l \times l$ matrixes over $GF[2]$. The proposed extractor is:

$$\mathsf{ext}_A : \{0,1\}^l \times \{0,1\}^l \mapsto \{0,1\}^n : (x,y) \mapsto \Big((A_1 x) \cdot y, (A_2 x) \cdot y, \ldots, (A_n x) \cdot y\Big),$$

where $\cdot$ is the inner product mod 2 and $A_i x$ is a matrix-vector multiplication over $GF[2]$. In practice, the matrices A_i can be specified in different ways, giving rise to different tradeoffs between the quality of the extraction and the implementation efficiency. For example, [3] mentions cyclic shift matrices, right shift matrices or matrices obtained from error correcting codes.

In the following, and for illustration (no particular constraints are imposed on the extractors in [4]), we will consider cyclic shift matrices [12]. That is, we just define A_i as the linear transformation matrix corresponding to a cyclic shift of $i-1$ bits, with l a prime having 2 as primitive root, *e.g.* for $l = 5$:

$$A_1 = \begin{pmatrix} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0 \end{pmatrix}, \quad \ldots \quad , A_5 = \begin{pmatrix} 0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \end{pmatrix}.$$

3 Implementation

In order to investigate the practical security provided by a stream cipher, it is needed to implement it in a reference device. As a case study (and because they are standard targets in side-channel attacks), we will consider an implementation in a small 8-bit microcontroller. For the AES Rijndael, we can rely on the implementation description provided in [2] for 8-bit processors. In this section, we discuss the implementation of the extractor in a similar device.

First note that, since we use a length tripling PRG based on the AES, the extractor in Figure 1 must generate 128 bits out of two sources of 192 bits. Hence, we will use $l = 193$ and pad one constant bit to the inputs of the extractor[2].

[2] This is necessary to meet the constraint that l has to be prime.

Second, the extractor applies to one public and one secret value. In order to minimize the amount of computations applied to the secret value, we will assume that, when computing $(A_i x) \cdot y$, x is public (hence corresponding to $X(i)$ in Figure 1) and y is private (hence corresponding to $S(i)$ in Figure 1).

Let us denote the cyclic shift of x by $i-1$ bits as $x_i = \mathsf{rot}(x, i-1)$. Implementing the extractor essentially corresponds to computing n inner products between 128 public x_i's and the secret y. In an 8-bit device, producing one bit b_i corresponding to one input x_i can be done as follows:

1. Denote the 16 bytes of x_i and y as x_i^j and y^j, respectively, $j \in [1;16]$.
2. Compute 16 bitwise AND operations: $z_i^j = x_i^j \odot y^j$,
3. Compute the bitwise XOR between the 16 z_i^j's: $w_i = \bigoplus_{j=1}^{16} z_i^j$,
4. Compute the XOR between the bits of w_i: $b_i = \bigoplus_{i=1}^{8} w_i(j)$,

where $w_i(j)$ denotes the jth bit of the word w_i. This process then has to be repeated 128 times in order to produce a PRG input (*i.e.* an AES key).

4 Attacking the leakage-resilient stream cipher

The stream cipher in Figure 1 is composed of a PRG and the previously described extractor. Looking at this construction from an adversarial point of view directly leads to the question: which part of the scheme is better to target? For this purpose, we will consider the resistance of these two parts of the construction against a standard DPA attack, such as formalized in [6]. As far as the PRG implementation is concerned, the situation is quite standard. Following the discussion about practical security in [11], we have a 3-limiting construction, meaning that an adversary is allowed to observe the leakage corresponding to the encryption of three different plaintexts (*i.e.* the constants c_1, c_2, c_3).

Choice of a leakage model and target implementation. In order to compare the security of the PRG to the one of the extractor with respect to side-channel attacks additionally requires to define a leakage model. In the following, we assume the so-called Hamming weight leakage model that is frequently considered to attack CMOS devices. It yields the implementation represented in Figure 2: for each byte of the public x_i and secret y, the adversary recovers the Hamming weight of their bitwise AND (possibly affected by some noise).

It is interesting to note that Figure 2 is very similar to a standard DPA targeting the key addition in the first round of a block cipher. In such a scenario, an adversary would use different plaintexts x_i and obtain the leakages corresponding to $l_i^j = \mathsf{WH}(x_i^j \oplus y^j)$. In the context of our extractor implementation, the only difference is that the bitwise XOR $\oplus$ is replaced by a bitwise AND $\odot$.

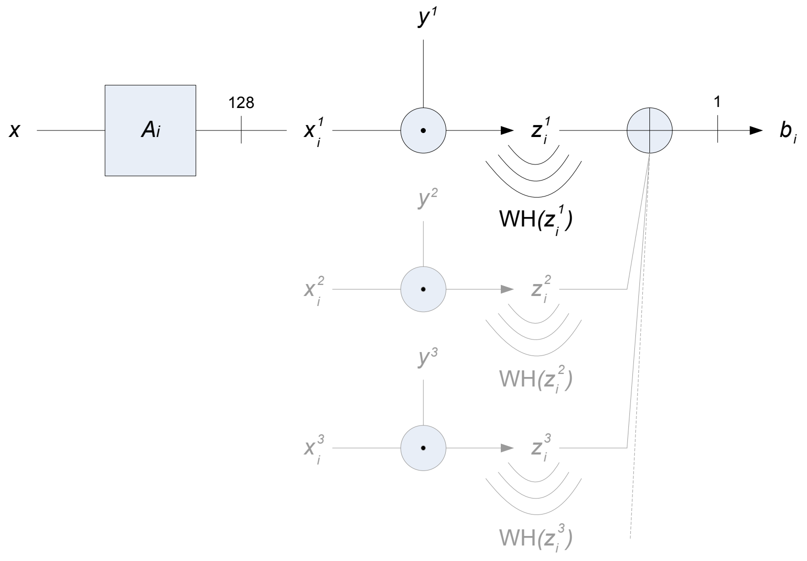

Fig. 2. Leaking implementation of the extractor.

Specificities of a DPA against the extractor. The main feature of a DPA against the implementation of Figure 2 is that the leakages do not have symmetry properties (discussed, *e.g.* in [9]), which leads to key dependencies that are rarely observed in side-channel attacks. A simple way to put forward these dependencies is to compute an information theoretic metric such as advocated in [10]. Namely, considering leakages of the form $l_i^j = \mathsf{WH}(x_i^j \odot y^j)$ and removing the i, j indices for clarity, we can compute the following conditional entropy:

$$\mathrm{H}[Y|L,X] = -\sum_{y\in\mathcal{Y}} \Pr[y] \sum_{x\in\mathcal{X}} \Pr[x] \int_l \Pr[l|y,x] \log_2 \Pr[y|l,x]\ dl,$$

that is, in our case: $\mathrm{H}[Y|\mathsf{WH}(Y \odot X), X]$. This quantity reflects the amount of information that is provided by the leakage (here in a known plaintext scenario).

Consider a standard DPA where the leakages equal $l_i^j = \mathsf{WH}(x_i^j \oplus y^j)$. Then, because the bitwise XOR is a group operation, we have that:

$$\mathrm{H}[Y|L,X] = -\sum_{x\in\mathcal{X}} \Pr[x] \int_l \Pr[l|y,x] \log_2 \Pr[y|l,x]\ dl, \text{ for a given } y, \qquad (1)$$

$$= -\sum_{y\in\mathcal{Y}} \Pr[y] \int_l \Pr[l|y,x] \log_2 \Pr[y|l,x]\ dl, \text{ for a given } x. \qquad (2)$$

In other words, on average over the public x's, all the secret y's are equally difficult to recover. And on average over the secret y's, all the public x's yield the same amount of information. Quite naturally, the situation becomes different when the bitwise XOR is replaced by a bitwise AND. As a simple example, imagine that one byte j of the public x_i is null. Then, observing the Hamming weight of $0 \odot y^j$ leaks no information at all about y^j. Clearly, the information leakage now depends on both x and y and Equations (1), (2) do not hold anymore.

In fact, one can show that the amount of information leakage in the implementation of Figure 2 depends on the Hamming weights of x and y, by computing:

$$\mathrm{H}_x[Y|L,X] = -\sum_{y\in\mathcal{Y}} \Pr[y] \int_l \Pr[l|y,x] \log_2 \Pr[y|l,x]\ dl,$$

$$\mathrm{H}_y[Y|L,X] = -\sum_{x\in\mathcal{X}} \Pr[x] \int_l \Pr[l|y,x] \log_2 \Pr[y|l,x]\ dl,$$

that is, the conditional entropies where either the x's or the y's are not uniformly distributed but fixed to an arbitrary value. By computing these quantities for Hamming weight leakages, we observe that they are equal for all x's and y's having the same Hamming weight. It yields the following vectors:

HW(x)	0	1	2	3	4	5	6	7	8
$\mathrm{H}_x[Y\|L,X]$	8.00	7.00	6.50	6.19	5.96	5.80	5.66	5.55	5.45

HW(y)	0	1	2	3	4	5	6	7	8
$\mathrm{H}_y[Y\|L,X]$	4.00	5.04	5.75	6.17	6.30	6.17	5.75	5.04	4.00

Intuitively, this means that, on average over the secret y's, the public x's with high Hamming weight yield more information. And on average over the public x's, the secret y's with extreme Hamming weights are easier to recover.

Experimental results. We now compare the security of the extractor with the one of the AES Rijndael. More specifically, the construction in Figure 1 is 3-limiting for the AES and 1-limiting for the extractor. The question we tackle is to know whether it is easier to attack 3 iterations of the AES versus a single iteration of the extractor, in the previously described setting.

Regarding the AES implementation, the typical target operations in a DPA are the first round key addition and S-box layers. Here, we will consider two different attacks. First, a basic (univariate) attack in which only the leakage of the S-box is exploited. Second, an improved (bivariate) attack in which the leakage of both the key addition and the S-box are exploited. In both cases, we use a Bayesian distinguisher (aka template attack) such as described in [1].

Regarding the implementation of the extractor in Figure 2, the key observation is that it re-uses each secret byte n times in order to produce n random bits. Considering the same adversary as for the AES, *i.e.* a template attack recovering the secret bytes of the extractor one by one, each secret byte can be identified by $n = 128$ bitwise AND computations, even if the construction is 1-limiting.

The success rates of our experiments (each averaged over 1000 independent experiments) evaluated in four different noise scenarios (from low noise: $\sigma_n = 0.5$ to high noise $\sigma_n = 4$), and assuming uniformly random inputs, are represented in Figure 3. For the AES implementation, the X axis represents a number of queries and the bold dots represent the success rates after 1, 2 and 3 queries, as allowed by the construction. For the extractor implementation, the X axis represents the number of elementary operations, that is upper bounded to 128 (*i.e.* a single execution). It yields the following observations:

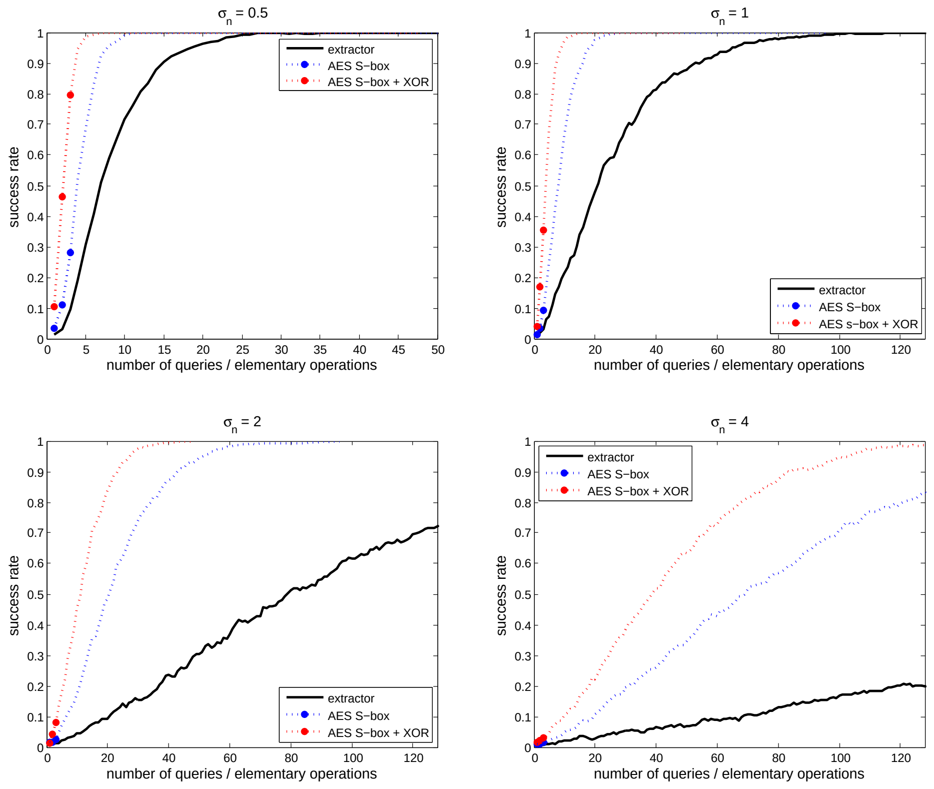

Fig. 3. Success rates of simulated experiments.

– The success rate of a 3-limited adversary against the AES implementation is anyway (much) smaller than the one of a 1-limited adversary against the extractor implementation. This answers the question in the beginning of this section. That is, an adversary who has to attack the PRG in Figure 1 implemented as described in this paper should focus on the extractor (rather than on the AES) in order to recover the secrets $S(i)$.

– Without considering the q-limit, the elementary operations of the extractor are "less informative" than the ones of the AES (as underlined by the previous information theoretic analysis). So, for a similar number of elementary operations observed (*e.g.* considering the AES S-box computations only, meaning one operation per query), the success rate against the AES implementation is higher than the one against the extractor.

5 Consequences

According to the results in the previous section, it is unlikely that any extractor will be able to provide high security levels (specially in small devices) without being combined with other (*e.g.* hardware-level) countermeasures. For example, in view of the implementation in Figure 2, the (time) randomization of the operations to execute could be done quite efficiently, following what has been achieved for the AES Rijndael in [5]. More generally, investigating the applicability of classical countermeasures such as masking and hiding to extractor implementations is an interesting scope for further research, and a necessary step to demonstrate the practical relevance of the FOCS 2008 construction.

Next to these general observations, we now investigate the possibility to propose more specific guidelines for the design of a leakage-resilient stream cipher using underlying principles similar to the ones described in [4].

Improving the extractor. Since it is necessary to protect the extractor implementation with countermeasures in order to guarantee a small enough λ-bit leakage, a natural guideline is to first consider low-complexity extractors. This is motivated both by the intuition that more computation generally give rise to more exploitable information [8] (although this is not a strict statement) and, maybe more importantly, by the need of efficient constructions that can run on a variety of low cost devices. In this respect, it is interesting to note that the majority of previous work in the area of randomness extraction focus on producing outputs as close to uniform as possible, much more than on implementation efficiency. So although constructions such as [3] are already quite efficient, there is probably room for further research in carefully designing extractors that are easy to implement and to protect against side-channel attacks.

Improving the construction. Since the extractor is actually the weak point in the implementation of Figure 2, another possibility is to modify the construction in order to impose a more challenging adversarial context when targeting the extractor with a DPA. For example, in Figure 1, the extractor takes two inputs: one public and one secret. This allows to mount a DPA in a known-plaintext scenario. But by applying the extractor to secret values only (possibly at the cost of some performance loss), the adversary would only be able to mount unknown-plaintext attacks. Interestingly, it is quite easy to quantify the impact of such a modification, in exactly the same setting as in the previous section. That is, we can now compute the following conditional entropy:

$$\mathrm{H}_y[Y|L] = -\sum_{x\in\mathcal{X}} \Pr[x] \int_l \Pr[l|y,x] \log_2 \Pr[y|l]\ dl,$$

where the input x is now kept secret. Doing this, the first consequence is that it is impossible to distinguish the secret x values having the same Hamming weight. The second consequence is that the information leakage is reduced as:

$\mathsf{HW}(y)$	0	1	2	3	4	5	6	7	8
$\mathrm{H}_y[Y\|L,X]$	4.67	6.38	7.29	7.76	7.92	7.84	7.53	7.03	6.33

6 Conclusions

This work first shows that the use of randomness extractors in physically observable cryptography can be paradoxical. On the one hand, they allow recovering (pseudo) entropy losses *if* the implementation of the extractor does not leak too much. On the other hand, the implementation of the extractor can become a better target for a DPA than an AES-based PRG, if no attention is paid. This observation makes a case for always discussing side-channel resistant primitives together with a clear specification of the algorithms they use and the devices on which they run. In particular, the selection of algorithms generally has an impact on both the security level that can be achieved assuming a bounded (so-called λ-bit) leakage per iteration *and* on the very value of the bound λ (as quantified by a success rate in our experiments). Problematically, not considering the overall impact of an algorithm and its implementation features in leakage-resilient cryptography may lead to inconsistencies in the resulting analysis. For example, the stream cipher of FOCS 2008 [4] has better security bounds than the one of Eurocrypt 2009 [7]. But in the practical analysis provided in this paper, the opposite conclusion holds (*i.e.* it is significantly easier to attack the FOCS 2008 construction in our setting). Admittedly, this observation is only based on a single case-study, and it should stimulate further research on the design and implementation of extractors in leakage-resilient cryptography, because of their interesting theoretical properties. However, we believe that our analysis, and the methodological conclusions that it brings, is reflective of a general situation and makes a case for reviewing several recent results in the field in light of a more practical security analysis, *e.g.* using the evaluation tools in [10].

We note again that these observations do not invalidate theoretical analyzes in physically observable cryptography but reduce their practical relevance. They mainly emphasize the need for interaction between formal proofs of physical security and low-level implementation issues. In the end, a useful construction needs to face the full complexity of the attacks, *i.e.* not only to assume small λ-bit leakages but also to find algorithms, and in the end, implementations, for which these small leakages can be obtained. In this respect, a first (and easy to manipulate) design criterion for leakage-resilient constructions would be to minimize the computational complexity of the algorithms that they exploit.

References

1. S. Chari, J. Rao, P. Rohatgi, *Template Attacks*, in the proceedings of CHES 2002, Lecture Notes in Computer Science, vol 2523, pp 13-28, CA, USA, August 2002.
2. J. Daemen, V. Rijmen, *The Design of Rijndael*, Springer, 2002.
3. Y. Dodis, A. Elbaz, R. Oliveira, R. Raz, *Improved Randomness Extraction from Two Independent Sources*, in the proceedings of APPROX-RANDOM 2004, pp 334-344, Cambridge, Massachussets, USA, August 2004.
4. S. Dziembowski, K. Pietrzak, *Leakage-Resilient Cryptography*, in the proceedings of FOCS 2008, pp 293-302, Washington, DC, USA, October 2008.
5. C. Herbst, E. Oswald, S. Mangard, *An AES Smart Card Implementation Resistant to Power Analysis Attacks*, in the proceedings of ACNS 2006, Lecture Notes in Computer Science, vol 3989, pp 239-252, Singapore, June 2006.
6. S. Mangard, E. Oswald, F.-X. Standaert, *One for All - All for One: Unifying Standard DPA Attacks*, Cryptology ePrint archive, report 2009/449.
7. K. Pietrzak, *A Leakage-Resilient Mode of Operation*, in the proceedings of Eurocrypt 2009, LNCS, vol 5479, pp 462-482, Cologne, Germany, April 2009.
8. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, *Algebraic Attacks on the AES: Why Time also Matters in DPA*, in the proceedings of CHES 2009, Lecture Notes in Computer Science, vol 5747, pp 97-111, Lausanne, Switzerland, September 2009.
9. W. Schindler, K. Lemke, C. Paar, *A Stochastic Model for Differential Side-Channel Cryptanalysis*, in the proceedings of CHES 2005, Lecture Notes in Computer Science, vol 3659, pp 30-46, Edinburgh, Scotland, September 2005.
10. F.-X. Standaert, T.G. Malkin, M. Yung, *A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks*, in the proceedings of Eurocrypt 2009, LNCS, vol 5479, pp 443-461, Cologne, Germany, April 2009, extended version available on the Cryptology ePrint Archive, Report 2006/139.
11. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, E. Oswald, *Leakage -Resilient Cryptography in Practice*, Cryptology ePrint Archive, report 2009/341.
12. U. Vazirani, *Efficient Considerations in Using Semi-Random Sources*, in the proceedings of STOC 1987, pp 160-168, New York, USA, May 1987.