Combining Leakage-Resilient PRFs and Shuffling

Towards Bounded Security for Small Embedded Devices

Vincent Grosso, Romain Poussier, François-Xavier Standaert, Lubos Gaspar,

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.

Abstract. Combining countermeasures is usually assumed to be the best way to protect embedded devices against side-channel attacks. These combinations are at least expected to increase the number of measurements of successful attacks to some reasonable extent, and at best to guarantee a bounded time complexity independent of the number of measurements. This latter guarantee, only possible in the context of leakage-resilient constructions, was only reached either for stateful (pseudo-random generator) constructions, or large parallel implementations so far. In this paper, we describe a first proposal of stateless (pseudo-random function) construction, for which we have strong hints that security bounded implementations are reachable under the constraints of small embedded devices. Our proposal essentially combines the well-known shuffling countermeasure with a tweaked pseudo-random function introduced at CHES 2012. We first detail is performances. Then we analyze it against standard differential power analysis and discuss the different parameters influencing its security bounds. Finally, we put forward that its implementation in 8-bit microcontrollers can provide a better security vs. performance tradeoff than state-of-the art (combinations of) countermeasures.

1 Introduction

Securing block cipher implementations in small embedded devices is a challenging problem. Popular countermeasures include masking, shuffling or the insertion of random delays, for which state-of-the-art solutions can be found in [3, 17, 21]. In practice, such countermeasures are usually combined in order to reach high security levels – which raises the question of which combinations bring the best security vs. efficiency tradeoffs. For example, mixing masking with shuffling has been shown to be an effective solution [9, 18], while mixing masking with random delays may be easier to cryptanalyze [5]. These type of combinations essentially aim at reducing the amount of information leakage per block cipher execution.

More recently, an orthogonal approach has attracted the attention of cryptographers, of which the goal is to limit the adversary's power by bounding its data complexity (i.e. number of plaintexts for which the leakage can be observed) or number of measurements. This approach is usually referred to as *re-keying* [11] or *leakage-resilient cryptography* [6] in the literature. From an application point-of-view, the most interesting primitives are stateless ones – Pseudo-Random Functions (PRFs), typically – since they provide essentially the same functionalities as block ciphers (so are useful, e.g. for encryption, authentication, hashing). In this context, the standard construction is the tree-based one from Goldreich, Goldwasser and Micali (GGM) [8]. Its leakage-resilience has first been analyzed under a random-oracle based assumption in [19]. A modified construction exploiting an "alternating structure" has then been proven secure in the standard model by Dodis and Pietrzak [4]. Faust et al. next succeeded to get rid of this alternating structure and to prove the GGM construction in the standard model, at the cost of some additional randomness requirements. Eventually, Yu Yu and Standaert showed how to relax these randomness requirements in minicrypt [22], and Abdalla et al. studied how to improve the efficiency of these constructions by exploiting a skip-list data structure [1]. All these previous results were obtained for non-adaptive leakage functions (i.e. pre-determined by the hardware).

In practice, the hope of leakage-resilient constructions is to obtain security-bounded implementations, in the sense that the time complexity of the best side-channel attack is lower-bounded, independent of the number of measurements performed by the adversary. Unfortunately, a recent report from Belaïd puts forward that this hope is not reached with the previously listed leakage-resilient PRFs[1]. This work further shows that the combination of leakage-resilient PRFs with masking does not lead to any significant security improvements. In this context, the only positive (heuristic) result of security bounded implementation was obtained for the tweaked PRF construction proposed by Medwed et al. at CHES 2012 [15], that takes advantage of hardware parallelism and carefully chosen plaintexts. The main idea of this tweak is to exploit plaintexts of the shape $p = (b, b, \ldots, b)$, i.e. where all the bytes entering the key additions are the same. Intuitively, if these bytes are manipulated in parallel, they create a "key-dependent algorithmic noise" that is hard to exploit by side-channel adversaries and may (under certain conditions) lead to security-bounded implementations. However, this positive result only applied to hardware implementations so far.

In this paper, we investigates whether more positive results can be obtained for software implementations, by combining the CHES 2012 tweaked PRF with the shuffling countermeasure. We next denote this proposal as the Shuffled PRF (SPRF) construction. The main motivation behind such a proposal is that key-dependent algorithmic noise can be produced by the parallel manipulation of carefully chosen plaintexts $p = (b, b, \ldots, b)$. Hence, since the impact of a shuffling is (under certain conditions) to emulate the noise of large parallel implementations within the constraints of small embedded devices, this combination could be effective. For this purpose, we first describe a framework allowing us to analyze the security of SPRF implementations against standard DPA attacks [13], and put forward that it depends on two main parameters: first, the amount of *direct leakage* on the S-box computations and permutation used for shuffling; second, the amount of *indirect leakage*, essentially due to the fact that the power

[1] In short, because these (stateless) PRF constructions can only bound the adversary's data complexity, by contrast with (stateful) leakage-resilient Pseudo-Random number Generators (PRGs) that bound the adversary's number of measurements.

consumed to compute different S-boxes may depend on the resource used and execution time. We then show that one type of indirect leakages (namely, different resources leaking differently at the same time sample) is beneficial to the adversary, while the other type (namely, the same resource leaking differently at different time samples) is detrimental. This suggests simple guidelines for cryptographic hardware designers willing to improve the security of SPRFs. We finally apply our results to the challenging case-study of an 8-bit microcontroller, and show that security-bounded implementations can be obtained under actual (direct and indirect) leakages. To the best of our knowledge, it is the first time that such a positive result is obtained for a small embedded device. Furthermore, and compared to the hardware construction in [15], our software scheme has the additional advantage that all operations (i.e. the key additions and S-boxes, but also MixColumns and the key scheduling) are shuffled with a 16-permutation. This mitigates possible weaknesses due to adversaries targeting one out of four MixColumns, hence reducing the key-dependent algorithmic noise.

2 Background

2.1 Leakage-resilient PRFs

We first describe the GGM construction evaluated on an input $x \in \{0,1\}^n$ under a key $k \in \{0,1\}^n$, next denoted $\mathsf{F}_k(x)$. It requires n stages and $2n$ random plaintexts p_b^i, with $b \in \{0,1\}$ and $1 \leq i \leq n$. Each stage consists in a block cipher execution, where an intermediate key k^i is computed from the previous intermediate key k^{i-1} and the plaintext $p^i_{x(i)}$ (i.e. $k^i = \mathsf{E}_{k^{i-1}}\big(p^i_{x(i)}\big)$, with $x(i)$ the ith bit of x, k^0 initialized to k and the output $\mathsf{F}_k(x)$ set to k^n. Taking AES as an example, this implies computing $n = 128$ block cipher executions to produce a single output. The tweak proposed in [15] is to use more (namely, 256) plaintexts of a specific shape per stage, leading to a total of 16 stages (plus one output whitening). For this purpose, the input is first split in 16 bytes denoted as $x = (x_1, x_2, \ldots, x_{16})$. Next and as illustrated in Figure 1, each stage updates the intermediate key as $k^{i+1} = \mathsf{E}_{k^i}(p_{x_{i+1}})$, with $1 < i \leq 16$ and plaintexts of a specific shape $p_{x_i} = (x_i, x_i, \ldots, x_i)$. Eventually, the output is defined as $\mathsf{F}_k(x) = \mathsf{E}_{k^{16}}(p)$ (with p an additional plaintext). Intuitively, the combination of a parallel implementation with the carefully selected plaintexts creates key-dependent algorithmic noise, and the output whitening prevents attacks exploiting the ciphertext (given that the block cipher is secure against SPA).

2.2 Shuffled AES implementation

Shuffling is a countermeasure against side-channel attacks that aims to randomize the execution of an algorithm over time. It has been applied to the AES in [9] and [21]. The main parameter influencing its security is the number of permutations randomizing each operation. Taking the simple example of the AES S-boxes, one can choose between executing them according to a random index

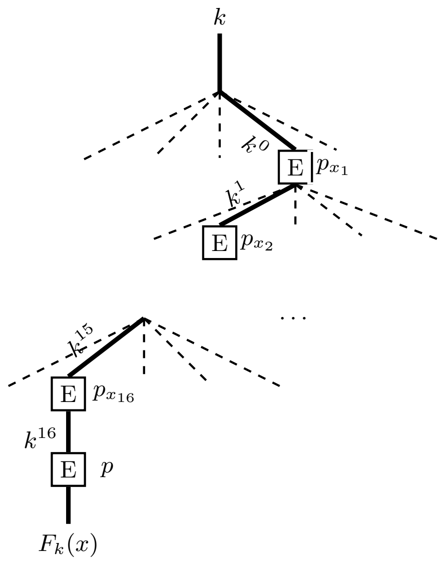

Fig. 1. Efficient leakage-resilient PRF.

(among 16 possible ones) that is just incremented, or according to a random permutation (among 16! possible ones). It has been shown in [21] that the first solution (although cheaper) may lead to very efficient attacks. In particular, an implementation protected with such a Random Start Index (RDI) may be as weak as an unprotected one for low noise levels. As a result, our following investigations will only consider shuffling based on a random permutation.

Besides the generation of a permutation vector (that is common to all solutions), different alternatives exist to implement a shuffled AES. The straightforward method requires an indirect indexing of the operands. That is, a counter is used to index a permutation vector, and the result is used to index the operand vector. Thus, instead of operating on registers directly, two RAM accesses are required for each (read or write) access to operands. This naturally leads to quite large cycle counts. A more efficient solution proposed at ASIACRYPT 2012 is to randomize the execution path. For this implementation, the assembly code of every transform is split into 16 independent blocks of instructions. Each of these 16 blocks is augmented with a label. This allows identifying its address in ROM. Furthermore, every transform is associated with an array of 17 16-bit words, where the first 16 words hold the addresses of the 16 blocks, and the 17th holds the address of the return instruction. During the execution of the cipher, the addresses are first re-ordered according to a previously generated permutation. Then, whenever entering a transform, a pointer is set to the beginning of the array in order to execute its 16 blocks of instructions in random order.

Table 1. SPRF performances compared to masking [17].

Implementation	Cycles count $\times 10^3$
First-order Masking	129
Second-order Masking	271
Third-order Masking	470
SPRF (double indexing)	788
SPRF (rand. exec. path)	252

Note that the implementations with double indexing and randomized execution path from [21] that we re-use in the following paid attention to shuffle *all* the AES operations with a 16-permutation. For this purpose, MixColumns is implemented in sets of 16 independent instructions based on xtime operations and three dummy key schedulings are interleaved with the real one (since the AES key scheduling only has four independent operations). Based on these choices, the cycle count of a SPRF implementation in an Atmel AVR micro-controller is given in Table 1, and compared with (1st-, 2nd- and 3rd-order) masking.

3 Evaluation framework

In order to analyze the security of SPRF implementations, we will use the standard DPA attacks defined in [13]. Furthermore, as our goal is to approach worst-case evaluations, we will consider the profiled setting of template attacks [2], and quantify their complexity with the security graphs described in [20].

Since the SPRF construction essentially relies on a shuffled AES design, the main challenge for our following investigations is to efficiently exploit the leakages of such implementations. In particular, and as previously discussed in [21] this requires to combine information obtained from the permuted operations in a meaningful way. As usual in side-channel attacks, we will target the first-round S-boxes and consider different (more or less ideal) models for this purpose.

Starting with the simplest situation, we can assume that all the S-boxes leak in the same manner (as in [15]), e.g. according to a Hamming weight function. This provides the adversary with 16-element vectors defined as:

$$\mathbf{L}_{\mathrm{u}} = [\mathsf{HW}(\mathsf{S}(x_0 \oplus k_0)) + N_0, \ldots, \mathsf{HW}(\mathsf{S}(x_{15} \oplus k_{15})) + N_{15}],$$

in the unprotected case, with N_i a Gaussian-distributed random noise with variance σ_n^2. Moving to a shuffled implementation, the vector becomes:

$$\mathbf{L}_{\mathrm{s}} = [\mathsf{HW}(\mathsf{S}(x_{\mathsf{p}(0)} \oplus k_{\mathsf{p}(0)})) + N_0, \ldots, \mathsf{HW}(\mathsf{S}(x_{\mathsf{p}(15)} \oplus k_{\mathsf{p}(15)})) + N_{15}],$$

with p the permutation used in the shuffling. Eventually, the SPRF construction will additionally force the same single value for all bytes, that is:

$$\mathbf{L}_{\mathrm{sprf}} = [\mathsf{HW}(\mathsf{S}(x \oplus k_{\mathsf{p}(0)})) + N_0, \ldots, \mathsf{HW}(\mathsf{S}(x \oplus k_{\mathsf{p}(15)})) + N_{15}].$$

In this context, the standard DPA adversary essentially requires a leakage model for the key byte s she targets, that can be written as:

$$\Pr[\mathbf{L} = \mathbf{l} | K_s = k] = \sum_t \frac{\mathsf{f}(t, s, \mathbf{l}')}{\sum_{t'} \mathsf{f}(t', s, \mathbf{l}')} \Pr[L_t = l_t | K_s = k],$$

where t is the time instant where an S-box is executed, $\mathbf{l}'$ is an optional vector containing information on the permutation p, L_t is an element of the leakage vector $\mathbf{L}$, and the function f indicates how the adversary deals with the permuted operations. In the (ideal) case where only the vector $\mathbf{L}_{\text{sprf}}$ would be available, the only possibility is to perform a direct template attack assuming a uniform prior for the permutation, i.e. $\mathsf{f}(t, s, \mathbf{l}') = 1/16$. Next, and based on a leakage model $\Pr[\mathbf{L}|K_s]$, the template adversary combines the leakage vectors corresponding to q different inputs for each candidate k_s using Bayes' law as follows:

$$p_{k_s} = \prod_{j=1}^{q} \Pr[k_s | \mathbf{L}^{(j)}, p^{(j)}].$$

For each target implementation in the next section we will repeat 100 experiments and for each value q in these experiments, use the rank estimation in [20] to evaluate the time complexity needed to recover the full AES master key. Eventually, we will build security graphs, where the attack probability of success is provided in function of a time complexity and number of measurements.

Incorporating indirect leakages. The execution of 16 S-boxes is illustrated in Figure 2 for unprotected and shuffled S-boxes. In this respect, one important observation made in [21] is the existence of *indirect leakages* on the permutation p, due to the fact that the different physical resources used to execute the S-boxes may leak according to different models. In order to capture this possibility in our simulations, we will define a family of linear leakage functions as:

$$\mathsf{L}_r(x) = \sum_{i=0}^{7} a_r^i \cdot x(i),$$

where the a_r^i are random coefficient within some interval (see next). These different leakage functions are directly reflected in the leakage vector as follows:

$$\mathbf{L}_{\text{sprf}}^{\text{r}} = [\mathsf{L}_{\mathsf{p}(0)}(\mathsf{S}(x \oplus k_{\mathsf{p}(0)})) + N_0, \ldots, \mathsf{L}_{\mathsf{p}(15)}(\mathsf{S}(x \oplus k_{\mathsf{p}(15)})) + N_{15}].$$

Intuitively, such resource-based indirect leakages break the assumption that all the S-boxes leak similarly, and help the adversary to know at which time instant a target S-box is executed. How strong are indirect leakages depends on the correlation between the models for different resources. In [15], FPGA experiments suggest that this correlation can rate between strong (i.e. 0.99 for S-boxes implemented in RAM) and weaker (i.e. 0.68 for combinatorial S-boxes).

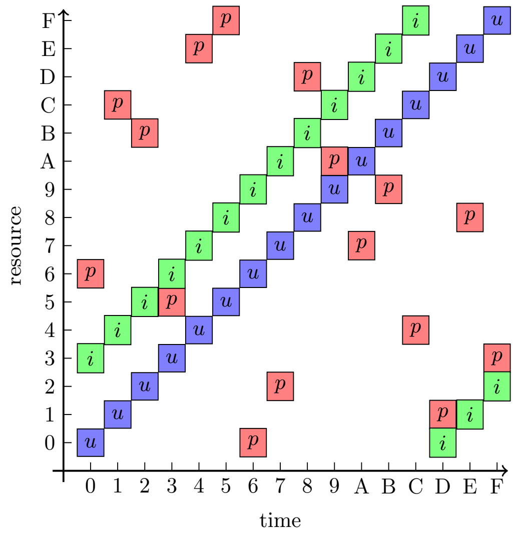

Fig. 2. S-boxes execution paths: unprotected device (u boxes, blue), shuffling with random start index (i boxes, green), shuffling with randomized permutation (p boxes, red).

Interestingly, we will show in the following that for SPRFs, the detrimental effect of leakage functions depending on the resource used is moderated by the fact that these functions may also depend on the time instant when they are executed (e.g. because of the pipeline state of a software implementation). We incorporate this possibility in our simulations with the following definition:

$$\mathsf{L}_{r,t}(x) = \sum_{i=0}^{7} a_{r,t}^{i} \cdot x(i),$$

which directly leads to the leakage vectors of the form:

$$\mathbf{L}_{\mathrm{sprf}}^{\mathrm{r+t}} = [\mathsf{L}_{\mathsf{p}(0),0}(\mathsf{S}(x \oplus k_{\mathsf{p}(0)})) + N_0, \ldots, \mathsf{L}_{\mathsf{p}(15),15}(\mathsf{S}(x \oplus k_{\mathsf{p}(15)})) + N_{15}].$$

Incorporating direct leakages. Quite naturally, indirect leakages are not the only information one can obtain about the permutation used for shuffling. If this permutation is generated on-chip, an informed adversary could also take advantage of a direct permutation leakage vector. In this context, the fact that the leakages functions depends on the resource used or time of execution has no impact on security. So we illustrate it with a Hamming weight function:

$$\mathbf{L}' = [\mathsf{HW}(\mathsf{p}(0)) + N_0, \ldots, \mathsf{HW}(\mathsf{p}(15)) + N_{15}].$$

Note however that direct permutation leakages can be avoided in certain cases, e.g. by randomizing the program memory as can be achieved (assuming a secure precomputation phase) within the recent FRAM technology [10].

4 Simulated experiments

Simulated experiments are convenient tools to evaluate implementations in various (more or less realistic) settings and to test the impact of different parameters on their security level. In the following, the first parameter we will play with is the amount of noise in the leakage vectors defined in the previous section. It is characterized by the variance of the noise variables N_i. In order to make its reading more intuitive, we will relate this noise level with the Signal-to-Noise Ratio (SNR), defined as the quotient between the variance of the mean leakage traces (aka signal) and the noise variance [12]. For Hamming weight leakages on 8-bit values, this signal equals 2 and we considered two noise levels for illustration: a weak one corresponding to $\sigma_n^2 = 0.1$, SNR= 20 and a stronger one corresponding to $\sigma_n^2 = 10$, SNR= 0.2. Based on these parameters, we first study the ideal case with no direct permutation leakage and all S-boxes leaking identically.

4.1 Ideal setting (identical S-box leakages, no direct perm. leakage)

In this case, the adversary is only provided with the leakage vector $\mathbf{L}_{\text{sprf}}$ and the only attack she can mount is a template one with uniform prior. As expected, the construction is security-bounded. That is, after a transient period, the template attack's time complexity saturates and becomes independent of the number of measurements. The impact of the noise parameter is clearly exhibited on Figure 3, where a higher noise level (i.e. 10 vs. 0.1) ensures a higher security bound (i.e. 2^{85} vs. 2^{60}) that is also reached for larger number of measurements (i.e. 1000 vs. 100). Note that these results even improve the ones of Medwed et al. [15] for hardware implementations, since a higher noise only implied a later saturation of the bound in this case (i.e. had no impact on the value of the bound).

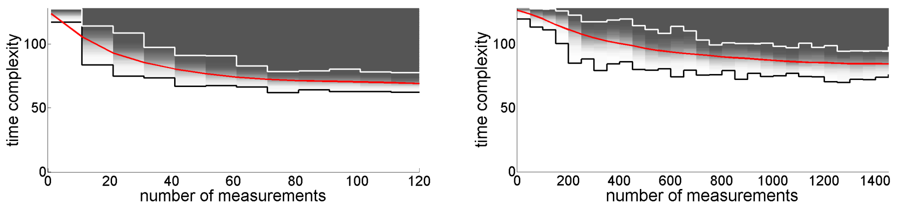

Fig. 3. Template attacks with uniform prior in the ideal scenario where all S-boxes leak identically. Left: low noise level ($\sigma^2 = 0.1$). Right: high noise level ($\sigma^2 = 10$).

4.2 Adding indirect resource-based leakages

We now move towards a more realistic scenario with indirect leakages due to the use of different resources in the implementation, helping the adversary to distinguish between the different S-boxes. That is, she can use the leakage vector

$\mathbf{L}^{\mathrm{r}}_{\mathrm{sprf}}$. Note that in this case, the only possibility remains to perform a template attack with uniform prior. But the probabilities $\Pr[L_t = l_t | K_s = k]$ now depend on the byte index $s = \mathsf{p}(t)$. This indirect information is directly obtained during profiling, so the attack methodology remains identical. As a result, the main additional parameter is the "similarity" of the leakage functions $\mathsf{L}_r(.)$ for different r's. For illustration, we will consider a high (average) correlation (of $\rho_r = 0.99$) and a smaller one (of $\rho_r = 0.75$). We picked up the leakage functions (more precisely, their coefficients a^i_r) randomly for our experiments, under the additional constraint that the signal was constant and set to 2, in order for the noise levels to have a similar meaning as in our previous Hamming weight based simulations. The results in Figure 4 clearly exhibit the weaknesses of the simulated SPRF implementations when the noise level is low and S-box leakages differ too significantly (e.g. for $\rho_r = 0.75$ in the left part of the figure) – they are not security-bounded anymore. Additional simulations performed at the higher noise level ($\sigma^2_n = 10$) are provided in Appendix A, Figure 12, and suggest that increasing the noise level is a simple way to preserve a security bound.

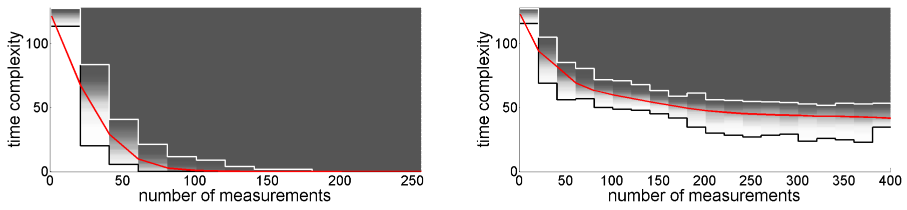

Fig. 4. Template attacks with uniform prior and indirect resource-based leakages, in the low noise scenario (i.e. $\sigma^2_n = 0.1$). Left: $\rho_r = 0.75$. Right: $\rho_r = 0.99$.

4.3 Mitigating resource-based leakages with time-based ones

We now consider the possibility to reduce the previous indirect information by making the leakage functions not only dependent on the resource used, but also on the time instant when they are executed. Intuitively, such dependencies are expected to make the exploitation of resource-based indirect leakages more difficult, by introducing some additional confusion between them due to the (useless for the adversary) time dependencies. In order to illustrate their impact, we stick with the most challenging scenario in the previous subsection, with low noise ($\sigma^2_n = 0.1$) and low similarity between the resources $\rho_r = 0.75$. We additionally consider weak and strong time-dependencies (with $\rho_t = 0.99$ and $\rho_t = 0.75$ for the leakage functions $\mathsf{L}_{r,t}(.)$, respectively). As illustrated in Figure 5, these time-dependencies indeed provide an efficient alternative way to reach security-bounded SPRF implementations, with lower noise levels (the same figure is provided for the high noise level in Appendix A, Figure 13.

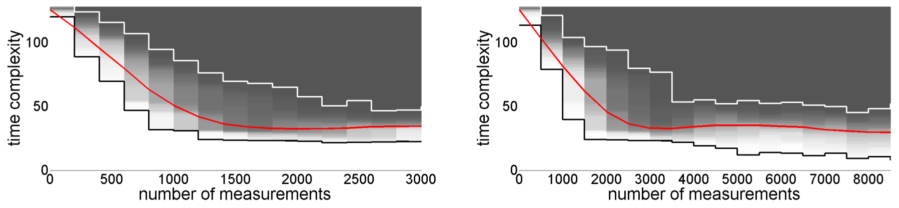

Fig. 5. Template attacks with uniform prior and indirect time+resource-based leakages (ρ_r=0.75), in the low noise scenario (i.e. σ_n^2=0.1). Left: ρ_t=0.99. Right: ρ_t=0.75.

Note that in this setting, the adversary has to estimate 16 × 16 templates, each of them corresponding to 256 intermediate values, which is a quite time-consuming task. Simplifying this profiling can result in a loss of informations.

4.4 Direct permutation leakage

Eventually and for completeness, we add the direct permutation leakage vector and consider an adversary who can exploit $\mathbf{L}_{\text{sprf}}^{\text{r+t}}$ and $\mathbf{L}'$. This context has been investigated in [21]: it requires an adversary performing a template attack with non-uniform prior and considers $\mathsf{f}(t, s, \mathbf{l}') = \Pr[L'_t = l'_t | K_t = K_t]$, where K_t is the part of the master key that is manipulated at time instant t. As in this previous work, we see that its impact on security is limited when the shuffling is based on random permutations – yet, they allow to converge faster towards the bound.

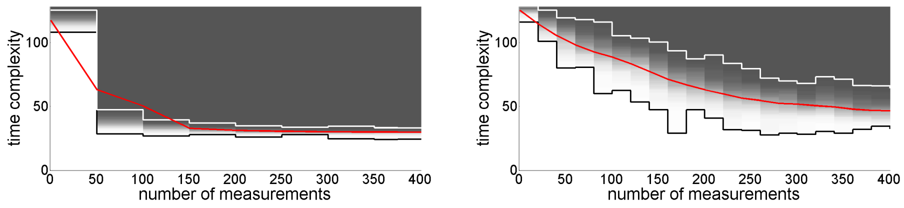

Fig. 6. Template attacks with direct and indirect (time+resource-based) leakages (ρ_r=0.75), in the low noise scenario (i.e. σ_n^2=0.1). Left: ρ_t=0.99. Right: ρ_t=0.75.

5 Practical experiments

The previous section suggests that SPRF implementations are promising candidates for designing security-bounded implementations in low-cost devices. It further puts forward that designers have two main parameters to increase their security level: the noise (as usual) and the time- vs. resource-based indirect leakages. In the latter case, we have strong incentive to design shuffled operations that only slightly depend on the resource used, and more significantly on their execution time. It naturally raises the question whether such designs exist in practice. In this respect, an interesting reference is the work on collision attacks in [7]: it shows that different implementations of the AES (e.g. always re-using the same registers or not) make the leakage models corresponding to different operations more or less similar (hence, collision attacks more or less realistic). We now provide an experimental case study based on an implementation of the SPRF construction in an Atmel AVR microcontroller. We first investigate the time- and resource-based dependencies in a shuffled AES implemented with double indexing and randomized execution path, then exhibit security evaluations based on these concrete values, and finally discuss scopes for further research.

In order to characterize the time- and resource-dependencies of the leakage models in our target AVR implementation, we build accurate templates for each S-box and time instant. As previously mentioned, this implies computing 16×16 sets of 256 templates – for each of them, we used 50,000 traces. Unfortunately, we rapidly found out that, both for the randomized execution path and the double indexing implementations, the time dependencies were small (i.e. with average values of $\hat{\rho}_t \approx 0.99$). By contrast, we could observe the quite strong resource-dependencies illustrated in Figure 7. Interestingly, we also noticed significant differences between the two approaches to shuffling. Namely, the double indexing implementation exhibits larger average values of $\hat{\rho}_r \approx 0.86$, compared to $\hat{\rho}_r \approx 0.5$ for the randomized execution path one. This intuitively matches the expectations for these two designs, since the first one is based on the repeated exploitation of a single register, while the randomized execution path inherently requires traveling through the different resources of the target device. In view of the performances listed in Table 1, this leads to a clear security vs. performance tradeoff.

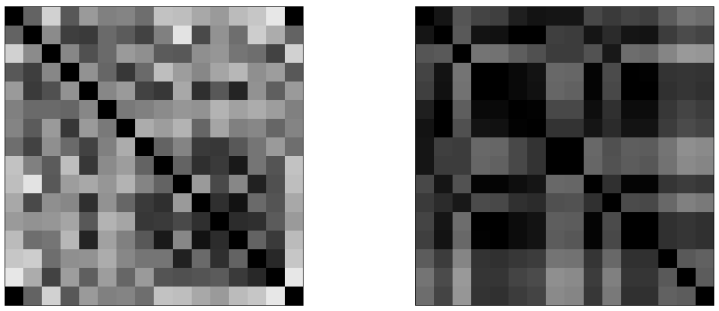
Fig. 7. Correlation between resources. Left: rand. exec. path. Right: double indexing.

We then launched experiments against these two implementations (with and without exploiting direct permutation leakages). In order to exhibit the impact of indirect leakages, we first analyzed an intermediate scenario, where the template mean values follow exactly the patterns of our target device, but we arranged the noise levels of all the leakage samples so that their SNR was fixed to a constant value. As expected and illustrated in Figures 8 and 9, the implementation based on double indexing allows a better security bound in this case.

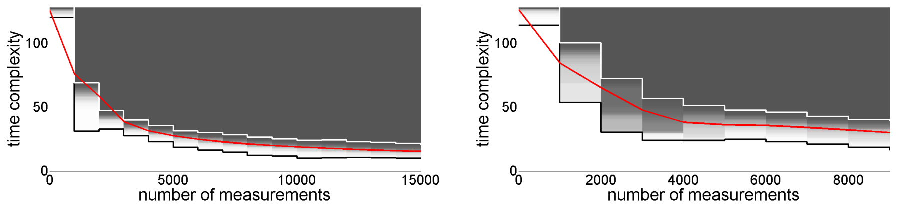

Fig. 8. Template attacks against the randomized execution path implementation ($\hat{\rho}_r$=0.5, $\hat{\rho}_t$=0.99, SNR=2). Left: with direct leakages. Right: without direct leakages.

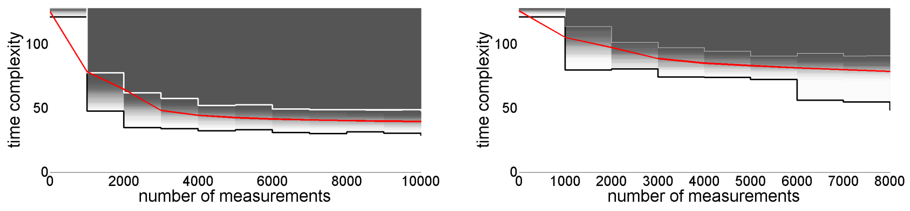

Fig. 9. Template attacks against the double indexing implementation ($\hat{\rho}_r$=0.86, $\hat{\rho}_t$=0.84, SNR=2). Left: with direct leakages. Right: without direct leakages.

We then considered the leakage samples with their actual noise level, as measured experimentally. It turned out (see Figures 10 and 11) that for the exploited samples, the SNR of the double indexing implementation was larger, hence canceling its advantage over the randomized execution path implementation. The exact reason of this observation is hard to state with confidence (we assume the additional memory manipulation of intermediate values in the double indexing implementation may be in cause). But this last experiment confirms the subtle dependencies between our two parameters on the concrete security level of an implementation. Since the leakage models are admittedly hard to control in cryptographic devices, this suggests that ensuring a large enough noise level may be the most reliable way to ensure large enough security levels in practice.

Discussion. The previous results are worth a couple of words of interpretation as we now detail. First, from a pragmatic complexity point-of-view, the values of the security bounds obtained may not be sufficient (as the enumeration of up to 2^{50} keys is reachable by determined adversaries and improved attacks and measurement strategies can probably be deployed). Yet, the very fact of being security-bounded is already a significant improvement compared to most state-of-the-art countermeasures (e.g. the combination of masking and shuffling).

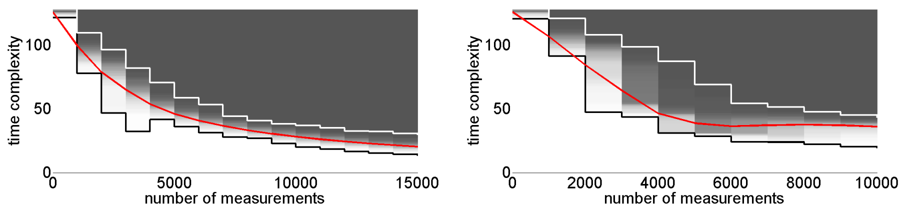

Fig. 10. Template attacks against the rand. execution path implementation ($\hat{\rho}_r$=0.5, $\hat{\rho}_t$=0.99, variable SNR). Left: with direct leakages. Right: without direct leakages.

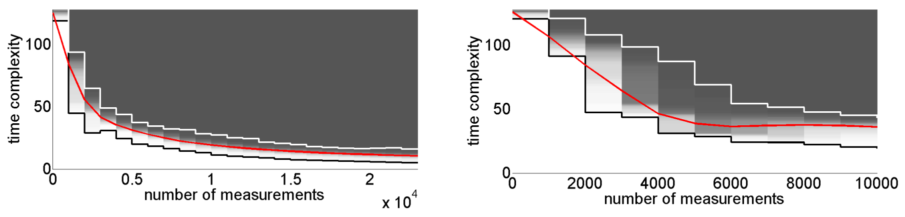

Fig. 11. Template attacks against the double indexing implementation ($\hat{\rho}_r$=0.86, $\hat{\rho}_t$=0.84, variable SNR). Left: with direct leakages. Right: without direct leakages.

Combined with the simulated results in the previous section, showing that it is possible to improve these bounds with higher noise or less informative indirect leakages, we believe this section confirms that SPRFs lead to an interesting family of protected implementations, that are certainly worth further investigation. In particular, we conjecture that combining it with a commercial security chip (including some hardware countermeasures) could already lead to much better concrete results. Furthermore, the best exploitation of time-dependent resource leakages is a nice research scope as well. In this respect, it is finally worth mentioning that the constructive investigation of the similarities between leakage models as we envision here is different (more demanding) than the destructive one in collisions attacks. That is, while a single sample showing good similarity is enough for these attacks to succeed, we need to guarantee that all of them are similar (for resource-based indirect leakages) or different (for time-based indirect leakages) – which also raises interesting characterization challenges.

Acknowledgements. F.-X. Standaert is a research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the European Commission through the ERC project 280141 (CRASH).

References

1. Michel Abdalla, Sonia Belaïd, and Pierre-Alain Fouque. Leakage-resilient symmetric encryption via re-keying. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *LNCS*, pages 471–488. Springer, 2013.

2. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.
3. Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In Mangard and Standaert [14], pages 95–109.
4. Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *LNCS*, pages 21–40. Springer, 2010.
5. François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient removal of random delays from embedded software implementations using hidden markov models. In Stefan Mangard, editor, *CARDIS*, volume 7771 of *LNCS*, pages 123–140. Springer, 2012.
6. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
7. Benoît Gérard and François-Xavier Standaert. Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version. *J. Cryptographic Engineering*, 3(1):45–58, 2013.
8. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479. IEEE Computer Society, 1984.
9. Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS*, volume 3989 of *LNCS*, pages 239–252, 2006.
10. Stéphanie Kerckhof, François-Xavier Standaert, and Eric Peeters. From new technologies to new solutions exploiting FRAM memories to enhance physical security. to appear in the proceedings of CARDIS 2013, LNCS, vol xxxx, pp yyy-zzz, Berlin, Germany, November 2013.
11. Paul C. Kocher. Leak resistant cryptographic indexed key update. US Patent 6539092.
12. Stefan Mangard. Hardware countermeasures against DPA ? a statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *LNCS*, pages 222–235. Springer, 2004.
13. Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
14. Stefan Mangard and François-Xavier Standaert, editors. *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *LNCS*. Springer, 2010.
15. Marcel Medwed, François-Xavier Standaert, and Antoine Joux. Towards super-exponential side-channel security with efficient leakage-resilient PRFs. In Prouff and Schaumont [16], pages 193–212.
16. Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *LNCS*. Springer, 2012.
17. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Mangard and Standaert [14], pages 413–427.
18. Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *LNCS*, pages 171–188. Springer, 2009.

19. François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer, 2010.
20. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 126–141. Springer, 2013.
21. Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *LNCS*, pages 740–757. Springer, 2012.
22. Yu Yu and François-Xavier Standaert. Practical leakage-resilient pseudorandom objects with minimum public randomness. In Ed Dawson, editor, *CT-RSA*, volume 7779 of *LNCS*, pages 223–238. Springer, 2013.

A Additional figures

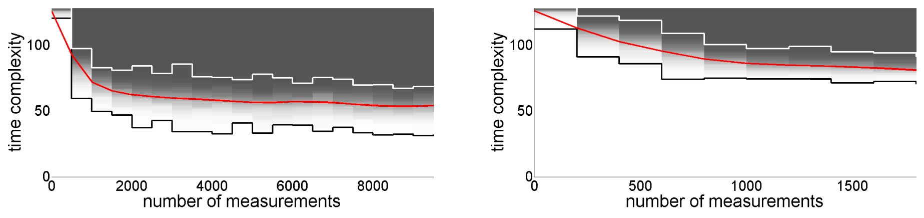

Fig. 12. Template attacks with uniform prior and indirect resource-based leakages, in the high noise scenario (i.e. $\sigma_n^2 = 10$). Left: $\rho_r = 0.75$. Right: $\rho_r = 0.99$.

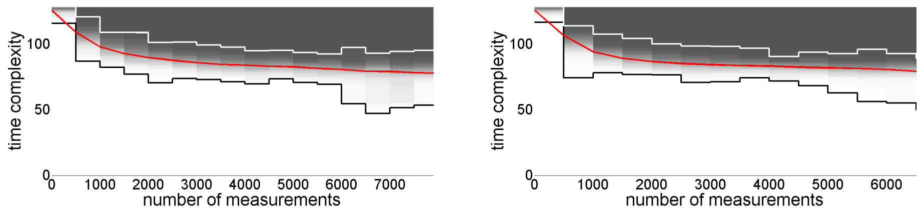

Fig. 13. Template attacks with uniform prior and indirect time+resource-based leakages (ρ_r=0.75), in the high noise scenario (i.e. σ_n^2=10). Left: ρ_t=0.99. Right: ρ_t=0.75.