Security Evaluations Beyond Computing Power

How to Analyze Side-Channel Attacks you Cannot Mount?

Nicolas Veyrat-Charvillon, Benoît Gérard, François-Xavier Standaert

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

Abstract. Present key sizes for symmetric cryptography are usually required to be at least 80-bit long for short-term protection, and 128-bit long for long-term protection. However, current tools for security evaluations against side-channel attacks do not provide a precise estimation of the remaining key strength after some leakage has been observed, e.g. in terms of number of candidates to test. This leads to an uncomfortable situation, where the security of an implementation can be anywhere between enumerable values (i.e. $2^{40} - 2^{50}$ key candidates to test) and the full key size (i.e. $2^{80} - 2^{128}$ key candidates to test). In this paper, we mitigate this important issue, and describe a key rank estimation algorithm that provides tight bounds for the security level of leaking cryptographic devices. As a result and for the first time, we are able to analyze the full complexity of "standard" (i.e. divide-and-conquer) side-channel attacks, in terms of their tradeoff between time, data and memory complexity.

Keywords: symmetric cryptography, concrete security, experimental cryptanalysis, divide & conquer attacks, side-channel analysis, leakage.

1 Introduction

Concrete security evaluations are at the core of cryptographic research. Taking the example of symmetric cryptography, they are at the same time central in formal definitions of security (e.g. as introduced by Bellare et al. [2]) and in the evaluation of attacks such as linear and differential cryptanalysis [4, 23]. Their goal is to provide bounds on the success probability of an adversary as a function of the resources she expends, typically measured in time, data and memory. But somewhat surprisingly, while such concrete (and complete) evaluations are usual in the context of mathematical cryptanalysis (e.g. [3], Table 5), they appear much harder to obtain in the context of physical cryptanalysis, even for "classical" attacks such as Kocher et al.'s Differential Power Analysis (DPA) [20].

The challenging nature of physical security evaluations mainly relates to the difficulty of capturing the "device-specificity" of the attacks. For example, statistical models used to evaluate the complexity of linear and differential cryptanalysis have been intensively studied for more than 20 years. They generally reflect the peculiarities of actual block ciphers to a good extent [10, 19, 30]. Under reasonable assumptions and using design tools such as the wide-trail strategy, one

can even guarantee security against large classes of statistical attacks [9]. By contrast, there is no general theory explaining how to build secure implementations, and most countermeasures used by device manufacturers highly depend on the chosen technology. Therefore, *present security evaluations against side-channel attacks need to rely on experimental validation.* For example, certification reports emitted by national authorities such as the ANSSI in France [1], or the BSI in Germany [6], are based on extensive analysis from evaluation laboratories.

From a cryptographic point of view, a purely empirical approach is hardly satisfying, as it only determines whether a given laboratory (with given equipment, time, data and memory) is able to recover some secret information contained in a leaking device. Hence, a fundamental question is to determine which parts of the physical security evaluations actually need experiments. Since the leakage in cryptographic implementations is technology-dependent, it is clear that some characterization through measurements is unavoidable, in order to determine its informativeness. Yet, given a certain amount of information leakage, it remains to analyze the impact of the time and memory complexities on the success probability of actual side-channel attacks. Answering these two questions (i.e. information extraction and exploitation) is the main goal of evaluation frameworks such as [31]. In the context of block ciphers (that will be our running example), most distinguishers published in the literature are based on a divide-and-conquer strategy[1]. As a result, the usual solution to exploit computational power is to perform enumeration [26, 32]. But *this implies that present security evaluations are limited to the computational power of the evaluator.* That is, the only leaking devices for which we can evaluate the security are the ones that are "practically insecure" (i.e. for which the leakage allows key enumeration). It leaves the (most interesting) evaluation of "practically secure" devices as an open problem. For example, an evaluation laboratory could claim that he could not recover an AES key within time complexity 2^{50}. But this does not give clear hints whether the security level of the target leaking device is 2^{51} or 2^{100}.

Main result. In this paper, we show that in the (realistic) scenario where the evaluator of a leaking device knows its secret key, it is possible to estimate the probability of success of "standard" side-channel attacks (e.g. the ones listed in footnote 1) that he is unable to perform (e.g. attacks of time complexities beyond 2^{80}). For this purpose, we provide a *rank estimation algorithm* solving the following problem: "given a set of discrete probability distributions for independent parts of a key, and a correct key k^*, provide tight bounds for the rank of this key among the set of all possible ones". Based on several experiments, we further show that our algorithm features small ratios between the lower and higher bounds on the key rank, and small running times (e.g. ratios between 2^2 and 2^{10} for a 128-bit key are obtained in a couple of seconds on a modern PC).

[1] Including Kocher et al.'s DPA, Brier et al.'s Correlation Power Analysis (CPA) [5], Chari et al.'s Template Attacks (TA) [8], Gierlichs et al.'s Mutual Information Analysis (MIA) [14], Schindler et al.'s stochastic approach [28], and many variations.

Consequences. Besides being a tool of choice for side-channel evaluation laboratories, our algorithm has a number of important consequences for the theory and practice of side-channel attacks. First, and for the first time, it allows the estimation of all the metrics put forward in [31] (namely, the success rates of all orders and guessing entropy). For example, the estimation of the guessing entropy for large master keys was previously impossible, as illustrated by the reports of the DPA contests `v1` and `v2` [27]. Our rank estimation algorithm could be directly used in further versions of such contests. Second, it provides a method to connect actual side-channel attacks with the need of "limited information leakage" in certain formal works aiming to prove security against physical attacks. For example, it allows quantifying the hardness assumption in Dodis et al.'s cryptography with auxiliary input [11], or the seed-preserving assumption used in [34]. Although less directly connected, it also provides a lower bound on the λ-bit leakage required in leakage-resilient cryptography [12]. Third, rank estimation yields an exact solution to evaluate the complexity of a number of "non-standard" side-channel attacks. For example, it would be perfectly suited to estimate the workload of collision attacks such as [25, 29] (see [13], Section 5). It would also be handy for analyzing the key-dependent algorithmic noise in the CHES 2012 leakage-resilient PRF [24], where most subkeys cannot be highly rated by the adversary. Fourth, our experiments suggest a cautionary note for the use of lightweight ciphers with small key sizes in leaking devices, as a few measurements can be enough to degrade their security within adversarial reach. Finally, we note that the proposed algorithm is not limited to physical security evaluations, and is potentially useful in any mathematical cryptanalysis setting where experiments are still needed to validate an hypothetical model.

2 Rank estimation: an overview

This section describes the approach that allows us to perform efficient and accurate rank estimation for standard key spaces (i.e. from 2^{80} to 2^{256}, typically).

Let us denote the independent parts of the key for which information has been obtained as subkeys. Our general idea is to organize the key space by sorting each of these subkeys according to their posterior likelihood, in decreasing order. As a result, the full key space is partitioned into 2 main volumes. The first one is defined by all key candidates with probability higher than the correct key. The second one is defined by all key candidates with probability smaller than the correct key[2]. Given this geometrical representation, our rank estimation problem can be stated as the one of finding bounds for the "higher" and "lower" volumes.

Organizing the key space with such volumes has one main advantage. Namely, the "higher" (resp. "lower") set of key candidates is delimited by a concave (resp. convex) surface within the key space. This means that if we pick a key candidate

[2] Between these volumes, we may find other volumes where all key candidates have the same probability as the correct one - which will be considered by our algorithm. Yet, in practice this "middle zone" usually contains the correct key only.

with a probability higher (resp. lower) than the target key, all keys with index lower (resp. higher) will also have the same property. This fact is illustrated in Figure 1 for a simplified 2-dimension case, with small subkey spaces. In this example, the correct key is the blue circle, and the equipotential surface splits the key space into candidates with higher (green, light) and lower (red, dark) probabilities. If one picks a key candidate within the lower probability set (e.g. the black circle), we notice that all candidates with higher indexes (inside the gray rectangle) will be on the same side of the surface. Hence, they will have a lower probability than that of the correct key. Taking advantage of this fact, our method for rank estimation essentially consists in "carving" such boxes of key candidates on each side of the probability surface, and use the volumes of these boxes to progressively refine the (lower and higher) bounds on the key rank.

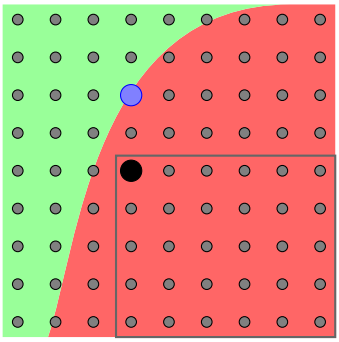

Fig. 1. Sorted key space in a simplified two-subkeys case.

Note that for small key spaces such as in Figure 1, standard quadrature tools could be used to bound the key rank. But as soon as typical cryptographic parameters are considered, e.g. 16 dimensions supported by 2^8 discrete points for the case of AES-128, such tools do not work anymore. Even when partially merging some dimensions[3] (e.g. in order to obtain 4 dimensions with support 2^{32}), such tools fail to provide an answer in reasonable time. To the best of our knowledge, the algorithm carefully described in the next section is the first one to provide an efficient solution that fits cryptographic evaluation purposes.

3 Algorithms for efficient rank estimation

This section aims at presenting the rank estimation algorithm. First, a high-level description of the algorithm is provided. For this algorithm to work efficiently in practice, several refinements are needed, detailed in the subsequent sections.

[3] Which will generally be beneficial to improve the performances of our approach too.

3.1 High-level algorithm representation

Algorithm 1 is a high-level view of the rank estimation algorithm. For the sake of simplicity, we assume that the target cipher is AES-128 (that is we are provided 16 subkey probability mass functions of support 2^8 each). The algorithm remains valid for other ciphers by just modifying the corresponding entries.

Algorithm 1: Rank estimation algorithm.

Input: Subkey distributions $\mathcal{D} = (D_i)_{1 \leq i \leq 16}$ and the key probability p^*.
Output: An interval $I = [I_0; I_1]$ containing the key rank.

```
L ← {[0; 255]^16};
I ← [0; 2^128];
PreProcess(D);
while L ≠ ∅ do
    (V, L) ← PickVolume(L);
    if IsCarved(V) = false then
        (V', I) ←CarveBox(V, I, p*, D);
        L ← InsertVolume(L, V');
    else
        {V_i'} ← SplitVolume(V);
        L ← InsertVolume(L, {V_i'});
return I;
```

As stated in Section 2, the algorithm is based on the fact that when distributions are sorted by decreasing probability, the frontier between the "lower" and "higher" key spaces is convex. Thus, a `PreProcess()` procedure is first used to sort distributions. It also performs other treatments in order to improve the algorithm efficiency (that will be discussed in Section 3.2). After initializing a list $\mathcal{L}$ with a volume containing the whole key space, we iterate over volumes in this list until it is depleted or the target accuracy is reached. The choice of the volume to consider is made by the `PickVolume()` procedure that removes the largest volume from $\mathcal{L}$. Since we are only interested in the largest volume, this list is efficiently implemented using an heap-based priority queue. The extracted volume is then processed. For reasons that will be clarified later in the section, we store volumes as either simple boxes (i.e. a Cartesian product of intervals), or the set difference of two such boxes that we will denote as carved volumes. Depending on the case, two alternative procedures are possible.

If the extracted volume is a full box, the `CarveBox()` procedure is called, that chooses a point on one side of the equipotential surface and carves a key set according to this point. The result is a carved volume V', which is actually a difference of two key boxes. The rank estimate I is updated according to the carved set. Determining the position of the point from the frontier requires the knowledge of the correct key probability p^* and the distributions $\mathcal{D}$. Afterwards, the

remaining carved volume is inserted back into list $\mathcal{L}$ using the `InsertVolume()` procedure. Else the volume extracted from list $\mathcal{L}$ is a carved volume, in which case the `SplitVolume()` procedure is used first, that splits it into smaller volumes having simpler geometries. As in the first case, these volumes are then inserted back into the list using the `InsertVolume()` procedure.

An run of the algorithm is illustrated in Figure 2. First, a box is carved from the key space and subtracted from the higher bound. The resulting carved box is then split in two. In the third step, a new box is carved on the green (light) side of the top box, and added to the lower bound. Finally, another box is carved from the green (light) side of the bottom box. After several additional steps, an exact bound can be given for the correct key in Figure 1. Note that during the rank estimation for an actual cipher, the limiting surface has too many details to be exactly computed, hence we are limited to an estimation of the rank.

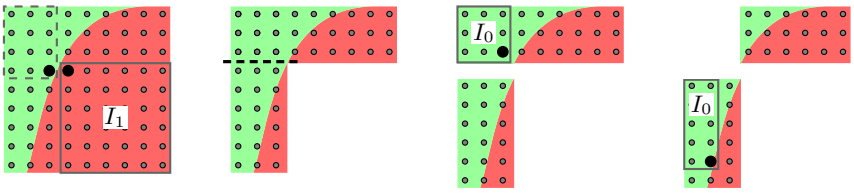

Fig. 2. Example run of Algorithm 1.

This algorithm is seemingly simple, but any direct implementation results in either intractable computation time or memory requirements. We only managed to attain tractability (and efficiency) through several specific choices and refinements for the different procedures, described in the remainder of the section.

3.2 The `PreProcess` procedure

In addition to sorting the distributions by decreasing probability, the `PreProcess` procedure also lowers the number of dimensions in the key space by merging some subkey lists. Instead of treating a 16-dimension cube of side length 2^8, we will work, for instance, on a 6-dimension space with sides up to 2^{24}. This merging step leads to significant improvements in the algorithm efficiency and should be applied as far as memory allows it. The impact of such merging on the algorithm performances will be illustrated in the experiments of Section 4.

3.3 The `PickVolume` procedure

When extracting a volume from the list, one can either pick the largest one or the smallest. In practice, extracting the smallest volume leads to small memory requirements, as we basically perform a depth-first exploration of the key space.

But this strategy has high computation time and becomes intractable as the algorithm gets lost in the details of the equipotential surface. By contrast, picking the largest volumes first is equivalent to performing a breadth-first search, and allows bounds to converge faster. The problem now is the increasing memory requirements (as many more volumes have to be stored at any given time). Hence, improvements were needed to minimize the memory cost, as described next.

3.4 The `InsertVolume` procedure

When the number of volumes stored in the queue increases beyond what we can efficiently store on a computer, we have two solutions: either switch to a depth-first search (which does stop the storage increase but has the side-effect of slowing down the convergence of bounds almost to a stop), or we can truncate the smallest volumes in the heap (since we use a heap and not a binary tree, we actually truncate volumes *among* the smallest ones, not exactly the smallest ones). The second approach naturally leads to accuracy losses in the estimation. Fortunately, we are not interested in the exact rank of the key but on "good enough" bounds. Hence, we opted for this second strategy. In practice, truncation is acceptable if the accuracy loss is small compared to the key rank, which depends on the storage limit set in the algorithm. We will show in the Section 4 that current computer memories allow very satisfactory results in this respect.

3.5 The `IsCarved` procedure

Iterations of Algorithm 1 essentially take boxes from the key space and carve other boxes out of them. If we were only able to represent plain boxes, we would need to perform splits along each dimension each time a box is carved. Essentially, carving a piece out of a box would lead to an increase in memory requirements (due to the storage of the resulting pieces): a naive split would generate up to $2^d - 1$ new boxes with d the number of dimensions. The storage technique suggested in Section 3.1 allows a significant reduction of this cost. By allowing the representation of *differences* between key boxes, we can store carved volumes within as much memory as "plain ones". As a result, the `IsCarved` procedure is used each time a volume is picked, in order to determine whether the volume passed as an argument is a plain box or a carved one.

3.6 The `SplitVolume` procedure

Whenever the volume we extract from the list is not a box, but rather a difference of two boxes, we have to simplify it and insert the resulting volumes back into the list. As stated above, the naive approach of splitting along each dimension is very inefficient and can generate up to $2^d - 1$ new boxes. Instead, we propose a slightly more complex way to do it. When given a volume consisting of a difference between two boxes, we split it along *a single* axis. This results in two volumes, one of which is a box, the other either a box or a carved volume. The

axis used to split is chosen so as to maximize the volume of the resulting simple box. This solution is illustrated in Figure 3. The carved box (left) can either be split into seven smaller boxes (middle), or into a larger box and another carved volume (right). We note that the latter approach, on top of minimizing the size of the volume list, preserves larger volumes. This additionally improves the refinement of bounds during the subsequent carving steps.

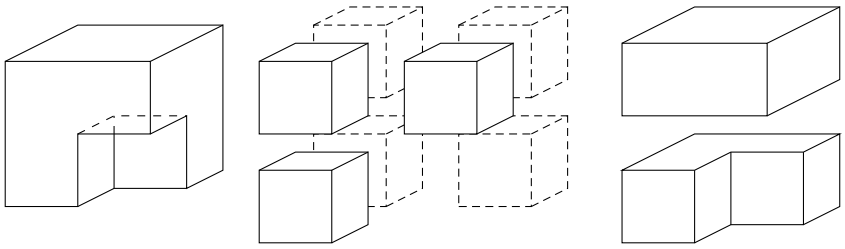

Fig. 3. Possible approaches to the volume split.

3.7 The `CarveBox` procedure

The carving point. In order to refine rank estimation, we classify key candidates inside a volume as more or less probable than the correct key, and carve pieces of this volume. The easiest way to do this is to pick the central point in the box, estimate its probability and carve the corresponding box. Unfortunately, this approach is rather inefficient as it will only classify one 2^{-d}-th of the box volume where d is the number of dimensions. Instead, we perform an heuristic optimization on the number of keys contained in the carved part, repeated on both sides of the equipotential surface. In practice, the surface is sufficiently "well-behaved" so that simple heuristics such as hill climbing (with some random restarts) suffice. More computationally intensive approaches (such as simulated annealing) only offer a marginal improvement while hampering speed.

The carving side. Iterations of Algorithm 1 provide two boxes for updating either the inner or outer bound of the key rank. In order to choose between them, we need a criteria. The naive proposal trying to minimize the *difference* between the updated bounds is not efficient, as it will almost always choose the biggest carved volume of the two. Indeed, the rank of the key in a side-channel attack usually tends to be small with respect to the size of the key space, with most of the keys having probability smaller than the correct one. As a result, such a criteria will almost always update the higher bound of the key rank and refine the lower one very late (i.e. when almost all key candidates have been classified). In order to avoid such shortcomings, a better criteria is to minimize the *ratio* (or log difference) between higher and lower bounds. This way, the carving is chosen so as to result in the fastest refinement of the bounds from a computational point of view - with typical bounds within a few orders of each other.

4 Experimental results

The goal of this section is two-fold. In a first part we evaluate the speed and accuracy of our rank estimation algorithm. In a second part we apply this tool in the context of a security evaluation and discuss its usefulness. Our experiments are based on a C++ implementation of the rank estimation algorithm. Its functional correctness was tested by comparing the results to those of an enumeration algorithm for computationally reachable key ranks.

4.1 Performances of the rank estimation algorithm

In this first part, we are only interested in gaging the efficiency of the rank estimation algorithm proposed in the previous section. To this end, we used for inputs the results of simulated template attacks against an unprotected AES-128 implementation. That is, we considered standard DPA as described in [21], targeting the 16 S-boxes in the first AES round and taking advantage of leakage samples of the shape: $l_i = \mathsf{HW}(\mathsf{S}[x_i \oplus k_i]) + n_i$, with HW the Hamming weight function, x_i (resp. k_i) the ith byte of the plaintext (resp. key), and n_i a Gaussian-distributed noise variable. As a result, we obtained attack outcomes in the form of 16 lists of 256 posterior probabilities. Note that our following analysis is quite independent of the exact type of side-channel attack implemented. As discussed in [32], key enumeration (hence, rank estimation) apply to any profiled or non-profiled DPA. In fact, the only important parameter for our performance evaluations is the rank of the correct key candidate that is suggested by the attack. In practice, we played with the noise variable and number of measurements in order to make this rank vary. The main criteria used to measure the efficiency of our algorithm are speed, memory and the tightness of the bounds.

In this context, a first task is the study of the impact of merging subkey lists during the `PreProcess` procedure. We analyzed convergence of the bounds obtained by our rank estimation algorithm as a function of the execution time in the following contexts: (*i*) No merging has been done on the subkey posterior distributions, i.e. the algorithm considers 16 dimensions of support 2^8, leading to 4 KB of memory requirements for the tables; (*ii*) Subkey lists were merged two by two, i.e. the algorithm considers 8 dimensions of support 2^{16}, leading to 524 KB of memory requirements for the tables; (*iii*) Subkey lists were been merged by three, i.e. the algorithm considers 5 dimensions of support 2^{24} and one of 2^8, leading to 83 MB of memory requirements for the tables. Merging further would require more than 4GB of memory and was not necessary in our context. As can be noticed in Figure 4, merging has a strongly positive impact on the performances of the algorithm performances and thus merging should always be performed up to the memory limit. The different experiments presented in the rest of this section were always obtained by merging lists as per (*iii*).

The next point of interest is the rate of convergence of the proposed algorithm. In Figure 5, the ratios between higher and lower bound on the estimated rank are plotted against the actual rank. More precisely, if the algorithm returns

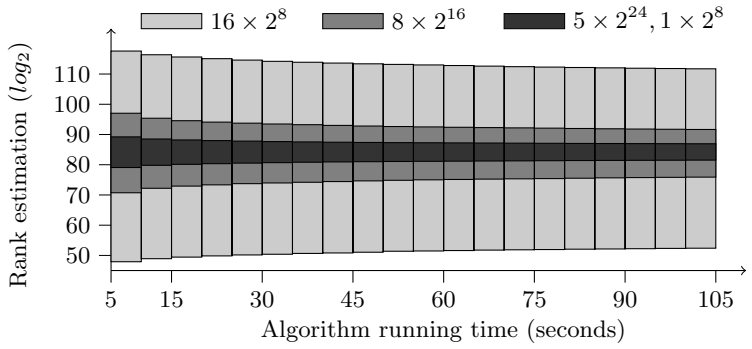

Fig. 4. Convergence of bounds versus running time depending on merge options.

an estimated interval $[i_0; i_1]$, then we plot $\log_2(i_1/i_0)$ on the Y-axis to show the relative accuracy of the current estimation, against the geometrical mean of the best estimated bounds on the X-axis. By repeating the experiment and removing outliers[4], we obtain the banana-shaped envelopes illustrated on the figure. Each envelope corresponds to a given running-time between 5 and 900 seconds.

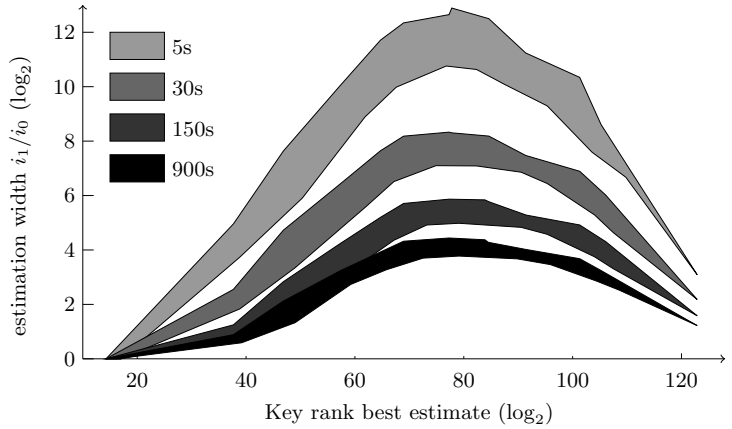

Fig. 5. Estimation accuracy versus rank (best estimate) given running time.

These results highlight that in our setting, most of the estimation work is done after 150 seconds. We also note that in 5 seconds, the ratio between higher and lower bounds is at most of 14 binary orders of magnitude, which is actually a very good indicator when evaluating the security of a cryptographic component. In this respect, ranks from 2^{60} to 2^{100} are the most difficult to estimate, while ranks smaller than 2^{30} can be accurately determined within a few seconds.

[4] Corresponding to cases where the equipotential surface is so simple that the algorithm returns a significantly more accurate interval compared to other experiments.

Eventually, another natural question is to determine how much our rank estimation algorithm allows improving crude lower bounds obtained by simply multiplying the subkey ranks together. Figure 6 shows the ratio between this approximation and the bounds returned by our algorithm after only 30s. Estimated intervals are plotted as gray vertical segments, with bound values divided by the rank product estimate. It can be observed that the product bound underestimates the actual rank by 20 to 40 binary orders of magnitude.

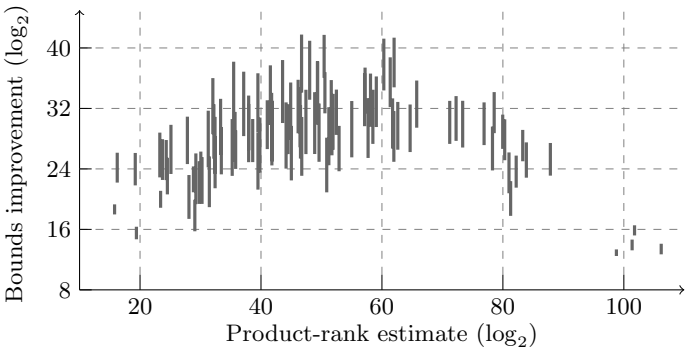

Fig. 6. Improvement obtained by rank estimation over rank-product lower bound.

4.2 Using rank estimation in physical security evaluations

In general, the main objective of a physical security evaluation is to determine how many encryption traces allow an adversary to recover the secret key with non-negligible probability, given some reasonable restrictions of its computing power and memory requirements. For this purpose, a natural solution is to estimate the metrics introduced in [31], namely the o-th order success rate (which gives the probability that the correct key stands among the o first ones provided by the attack), and the guessing entropy (which is the expected rank of the correct key after the attack). However, and as already mentioned in introduction, the estimation of these metrics was so far limited to subkeys, or to success rates of enumerable orders for master keys. In the remainder of this section, we show how efficient rank estimation can be used to mitigate this limitation, and provide the complete picture for side-channel attack security evaluations.

For this purpose, we define an "all-order" success rate graph, which provides the probability of a successful key recovery (on the Z axis or color map), depending on both the number of traces (on the X-axis) and the enumeration effort (on the Y-axis). That is, any point in this graph corresponds to the success rate for a given number of queries $q = x$ and order $o = y$. In order to relate this graph with the evaluation framework proposed in [31], we first remark that any of its "slices" obtained for a fixed y corresponds to a y-th order success rate. Furthermore, any slice for a fixed x is a rank distribution graph for a given number of measurements, the mean value of which equals the guessing entropy.

Building such a graph simply requires to perform several rank estimations, for different values of the number of encryption traces measured. These data provide a set of intervals that can then be used as input to a kernel density estimation (e.g. with uniform kernels). What is actually of interest to an evaluator is the cumulant of the density function resulting from this estimation: it indicates the probability that a key will be found, depending of the amount of keys that the adversary can enumerate. Interestingly, such density estimations converge quickly. For example, the security graphs in Figures 7, 8 were obtained by performing 100 attacks and estimating the key rank during 5 seconds, which amounts to less than 10 minutes of computation for each possible number of traces. The continuous black (resp. white) lines indicate the minimum (resp. maximum) ranks observed. Basically, any data/enumeration point below the black line appears safe, while points above the white line lead to certain key recovery. The medium zone indicates a non-negligible probability of key recovery.

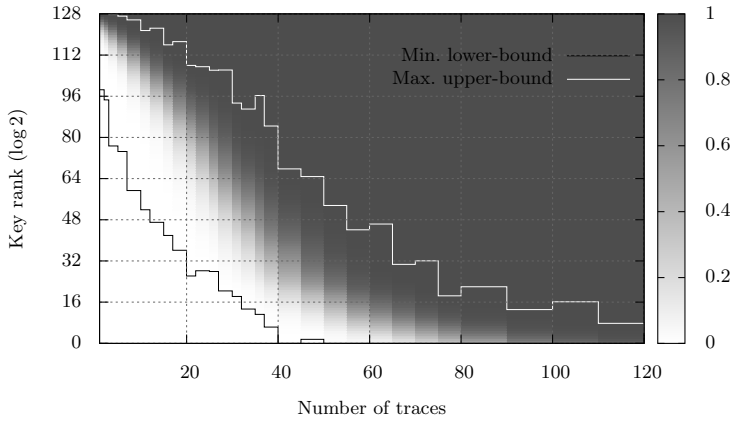

Fig. 7. Example of a security graph for a template attack against unprotected AES.

For illustration, we produced the security graphs for implementations of the block ciphers AES and LED [16], that have respective key lengths of 128 and 64 bits. Our experiments exploited exactly the same leakage models as in the previous section, with the same noise variance for both ciphers, and the number of target subkey bytes as only difference. In the case of the AES implementation shown in Figure 7, an adversary with a personal computer spending two weeks of computation (i.e. enumerating $\approx 2^{40}$ keys) will have a small chance of recovering the master key when she has less than 15 traces at her disposal, and will almost certainly succeed if she has more than 60 traces. Comparing this result with the classical (first-order) success rate curve, we observe that this success rate is still stuck at zero for 60 traces, hence suggesting the interest of key ranking in security evaluations. Besides, it is also interesting to observe that the impact of side-channel leakage is more critical for LED, as a small number of traces (e.g.

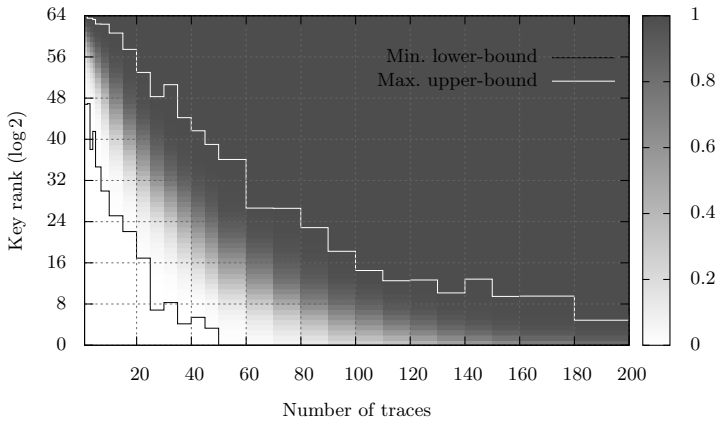

Fig. 8. Example of a security graph for a template attack against unprotected LED.

5 in the example of Figure 8) already allows decreasing the computing power required for key recovery down to approximately 2^{40} (whereas it is still around 2^{80} for the AES given the same number of traces). In general, the graphs of Figures 7 and 8 can be used directly to determine the re-keying rate in a leakage-resilient construction. The designer just has to choose an enumeration threshold corresponding to the computing power of the adversary considered, then he extracts the number of measurements that can be tolerated without risking a key recovery (using the figures black line minus a small security margin). Note that the rank and related enumeration effort can be translated into both time and memory costs thanks to the enumeration algorithm presented in [32]. Exemplary performances are reported in Table 1 (where enumeration times do not include the key testing as it depends on the cipher under attack). Figures 7 and 8 thus show the security of the device for different data/time/memory trade-offs.

		2^{20}	2^{30}	2^{40}	2^{50}	2^{60}
Enumeration (using [32])	Time	0.25 s	6 m	2 w	165 y	77×10^5 y
	Memory	<3 MB	100 MB	11.5 GB	1.35 TB	157 TB

Table 1. Time and memory requirements for key enumeration.

Eventually, and in order to confirm that the tools introduced in this paper apply to actual implementations as easily as to simulated experiments, we built the security graph corresponding the best attack submitted to the DPA contest v2 [27], depicted in Figure 9. Producing this graph required approximately 20 minutes of computation on an 8-core computer, and did not imply any modification of the rank estimation algorithm. Note that this evaluation was performed on the public database of the contest, which is easier to attack than private one. This explain the small discrepancy between our graph and the "Hall of Fame"

available online. Namely, the best attack reported in the contest needs 1173 traces to reach an 80% key recovery success rate when no enumeration is done. By adding enumeration up to rank 2^{32}, the data complexity requirement falls down to only 439 traces! In the "easier" context of the public database (described in Figure 9), the security graph shows that only 350 traces are needed with a 2^{32}-key enumeration, while a first-order success rate reaches 80% for 935 traces. To conclude, we mention that just as our rank estimation applies to profiled and non-profiled DPA, it also applies to protected implementations (e.g. with masking [7, 15] or shuffling [17, 22]). The main reason is that standard side-channel attacks against such implementations would produce lists of subkey scores or probabilities, as described beforehand and exploited in this paper.

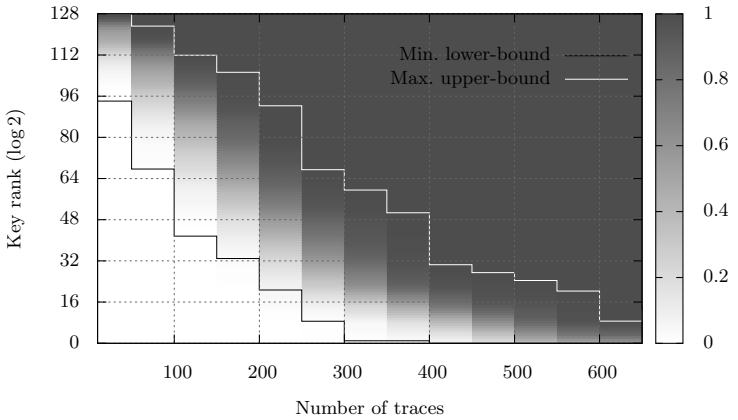

Fig. 9. Security graph for the latest attack of DPA contest v2 (public traces).

References

1. ANSSI. Agence nationale de la securite des systemes d'information, http://www.ssi.gouv.fr/en/products/certified-products/, retrieved on aug. 1, 2012.
2. Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403. IEEE Computer Society, 1997.
3. Eli Biham, Orr Dunkelman, and Nathan Keller. New results on boomerang and rectangle attacks. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.
4. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.
5. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Joye and Quisquater [18], pages 16–29.
6. BSI. Federal office for information security, https://www.bsi.bund.de/en/topics/certification/certification_node.html, retrieved on aug. 1, 2012.

7. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Wiener [33], pages 398–412.
8. 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 *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
9. Joan Daemen and Vincent Rijmen. The wide trail design strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.
10. Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *Journal of Mathematical Cryptology*, 1(3):221–242, 2007.
11. Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In Michael Mitzenmacher, editor, *STOC*, pages 621–630. ACM, 2009.
12. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
13. Benoît Gérard and François-Xavier Standaert. Unified and optimized linear collision attacks and their application in a non-profiled setting. *to appear in the proceedings of CHES 2012*, 2012.
14. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
15. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
16. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
17. 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 *Lecture Notes in Computer Science*, pages 239–252, 2006.
18. Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*. Springer, 2004.
19. Pascal Junod. On the complexity of Matsui's attack. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2001.
20. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener [33], pages 388–397.
21. S. Mangard, E. Oswald, and F.-X. Standaert. One for all – all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
22. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
23. Mitsuru Matsui. Linear cryptoanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

24. Marcel Medwed and François-Xavier Standaert. Towards super-exponential side-channel security with efficient leakage-resilient PRFs, non-profiled setting. *to appear in the proceedings of CHES 2012*, 2012.
25. Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2010.
26. Jing Pan, Jasper G. J. van Woudenberg, Jerry den Hartog, and Marc F. Witteman. Improving DPA by peak distribution analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 241–261. Springer, 2010.
27. Telecom ParisTech. http://www.dpacontest.org/, retrieved on aug. 1, 2012.
28. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.
29. Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: Combining side channel- and differential-attack. In Joye and Quisquater [18], pages 163–175.
30. Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *J. Cryptology*, 21(1):131–147, 2008.
31. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.
32. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. *to appear in the proceedings of SAC 2012*, 2012.
33. Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.
34. Yu Yu, François-Xavier Standaert, Olivier Pereira, and Moti Yung. Practical leakage-resilient pseudorandom generators. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 141–151. ACM, 2010.