An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks

Nicolas Veyrat-Charvillon, Benoît Gérard,
Mathieu Renauld, and François-Xavier Standaert

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

Abstract. Methods for enumerating cryptographic keys based on partial information obtained on key bytes are important tools in cryptanalysis. This paper discusses two contributions related to the practical application and algorithmic improvement of such tools. On the one hand, we observe that the evaluation of leaking devices is generally based on distinguishers with very limited computational cost, such as Kocher's Differential Power Analysis. By contrast, classical cryptanalysis usually considers large computational costs (e.g. beyond 2^{80} for present ciphers). Trying to bridge this gap, we show that allowing side-channel adversaries some computing power has major consequences for the security of leaking devices. For this purpose, we first propose a Bayesian extension of non-profiled side-channel attacks that allows us to rate key candidates according to their respective probabilities. Then we provide a new deterministic algorithm that allows us to optimally enumerate key candidates from any number of (possibly redundant) lists of any size, given that the subkey information is provided as probabilities, at the cost of limited (practically tractable) memory requirements. Finally, we investigate the impact of key enumeration taking advantage of this Bayesian formulation, and quantify the resulting reduction in the data complexity of various side-channel attacks.

1 Introduction

Side-channel attacks represent an important threat to the security of cryptographic hardware products. As a consequence, evaluating the information leakage of microelectronic circuits has become an important part in the certification of secure devices. Most of the tools/attacks that have been published in this purpose are based on a so-called "divide-and-conquer" approach. That is, in a first "divide" part, the evaluator/adversary recovers information about different parts of the master key, usually denoted as subkeys (as a typical example, the target can be the 16 AES key bytes). Next, a "conquer" part aims to combine the information gathered in an efficient way, in order to recover the full master key.

Research over the last ten years has been intensive in the optimization of the divide part of attacks. Kocher et al.'s Differential Power Analysis (DPA) [14] and Brier et al.'s Correlation Power Analysis (CPA) [6] are notorious examples.

One limitation of such approaches is their somewhat heuristic nature, as they essentially rank the subkeys according to scores that do not have a probabilistic meaning. As demonstrated by Junod in the context of linear cryptanalysis, such heuristic key ranking procedures may be suboptimal compared to Bayesian key recoveries [12]. The template attacks introduced by Chari et al. in 2002 typically aim to get rid of this limitation [7]. By carefully profiling a probabilistic model for the physical leakages, such attacks offer a direct path towards Bayesian subkey testing procedures. Template attacks are optimal from an information theoretic point of view, which makes them a prime tool for the worst-case security evaluation of leaking devices [32]. However, they also correspond to strong adversarial assumptions that may not be met in practice. Namely, actual adversaries are not always able to profile an accurate enleakage model, either because of a lack of knowledge of the target devices or because of physical variability [26]. As a consequence, attacks profiling an "on-the-fly" leakage model such as the stochastic approach, introduced by Schindler et al. [27] and discussed by Doget et pal. [9], are an important complement to a worst-case security analysis.

By contrast, only little attention has been paid to the conquer part in side-channel analysis. That is, in most cases the attacks are considered successful if all the subkeys are recovered with high confidence, which generally implies an extremely small time complexity for the offline computations. This situation is typically exemplified by initiatives such as the DPA contest [22], where the success rate in recovering a master key is directly obtained as the success rates for the concatenated 16 subkeys ranked first. In fact, the most noticeable exceptions attempting to better exploit computational power in physical attacks are based on advanced techniques, e.g. exploiting the detection of collisions [4,15,28,29], or taking advantage of algebraic cryptanalysis [5,20,24,25], of which the practical relevance remains an open question (because of stronger assumptions). But as again suggested by previous works in statistical cryptanalysis, optimal key ranking procedures would be a more direct approach in order to better trade data and time complexities in "standard" side-channel attacks.

In this paper, we improve the divide and the conquer parts of side-channel attacks, with two main contributions. Regarding the divide part, we start with the observation that non-profiled side-channel attacks are usually limited by their heuristic use of scores when ranking subkey candidates. As a consequence, we propose a method for non-profiled attacks that allows deriving probability mass functions for the subkey hypotheses. This tool can be viewed as a natural extension of the stochastic approach, but is also applicable to DPA and CPA. More generally, expressing the information obtained through non-profiled side-channel attacks with probabilities allows us to connect them better with template attacks, where the scores are already expressed as subkey probabilities.

Second, we provide the first comprehensive investigation of the conquer part of side-channel attacks. For this purpose, we start from the motivation that testing several billions of key candidates on a modern computer is not far-fetched: being able to recover a master key after such a computation is indeed a security breach. Next, we observe that two main solutions for testing key candidates from partial information on the subkeys exist in the literature. On the one hand, Meier and Staffelbach proposed a sampling algorithm in 1991. However, it turns out that in our side-channel attack context, such a probabilistic search leads to significant overheads in terms of number of keys to test. On the other hand, Pan, van Woudenberg, den Hartog and Witteman described a deterministic key enumeration algorithm at SAC 2010. But large memory requirements prevent the application of this second solution when the number of keys to enumerate increases. For example in [21], the authors were limited to the enumeration of 2^{16} candidates. As none of these tools is perfectly suited to our side-channel attack context, we propose a new deterministic algorithm for key enumeration, that is time and memory efficient, and allows the optimal enumeration of full keys by decreasing order of probabilities. It takes advantage of the probability mass functions of subkeys made available through our first contribution. The new algorithm can be viewed as an improvement of the SAC 2010 one, in which we reduce the memory complexity of the enumeration thanks to a recursive decomposition of the problem. Interestingly, and as previously observed in other cryptanalysis settings, this improvement of the key ranking strategy also has a significant impact on the data complexity of side-channel key recoveries.

Summarizing, this work brings an interesting complement to the evaluation framework in [32]. It allows stating standard side-channel attacks as a data complexity vs. time complexity tradeoff. On the theoretical side, the proposed key enumeration algorithm leads to a proper estimation of security metrics such as high-order success rates or guessing entropy, for block cipher master keys (this estimation was previously limited to subkeys or small orders). In practice, experimental results also exhibit that considering adversaries with a reasonable computing power leads to significant improvements of standard side-channel attacks. These gains are particularly interesting if we compare them with the ones obtained by only working on the statistics in the divide part of side-channel attacks [31]. Hence, we believe that the tools introduced in this paper have an important impact for the security evaluations of leaking devices, e.g. for certification laboratories. In this respect, it is worth noticing the gap between the computational complexities usually considered in the evaluation of side-channel attacks and the ones considered for evaluating security against mathematical attacks [16,23]. We also note that the tools introduced in this paper are generic and have potential impact in other cryptanalytic settings (e.g. based on faults [3], or statistical [2,18]), although standard side-channel attacks are a very natural environment for using them. Finally, in order to stimulate authors to consider the computational aspect of side-channel attacks, we provide an optimized implementation of the enumeration algorithm available in [1].

2 Background

The “standard” side-channel attacks considered in this work use a divide-and-conquer approach in which the side-channel subkey recovery phase focuses on one specific operation at a time [17]. In block ciphers like the AES, this operation is usually one 8-bit S-box in the first encryption round. We denote with p and k the byte of the plaintext and the byte of the key (i.e. the subkey) that are used in the attack, with $x = p \oplus k$ the input value of the S-box, and with $y = \mathsf{S}(x)$ the corresponding output value. The goal of a side-channel subkey recovery phase is to identify the best (and hopefully correct) subkey candidate $\hat{k}$ from the set of all possible key hypotheses $\mathcal{K}$, using q measured encryptions. For each target S-box the adversary collects a data set of pairs $\{(p_i, l_i)\}_{1 \leq i \leq q}$[1], with p_i the i^{th} plaintext byte involved in the target S-box computation, and l_i the corresponding leakage value. For simplicity, and because it has little impact on our following discussions, we assume unidimensional leakages. In addition, we assume leakage samples composed of a deterministic and a random part, with the deterministic part depending only on the S-box output (i.e. we use the EIS assumption introduced in [27]). The leakage samples can consequently be written as $l_i = \mathsf{L}(y_i) = \mathsf{f}(y_i) + n$, with n a Gaussian distributed noise. In general, side-channel attacks can be classified as profiled and non-profiled attacks, depending on whether the adversary can perform a training phase or not.

Profiled attacks, like template attacks [7], take advantage of their profiling phase to characterize the leaking device with a probabilistic model. This allows the adversary to rank the subkey hypotheses k according to their actual probabilities: $\hat{k} = \arg\max_k \Pr[k|\{(p_i, l_i)\}]$. These probabilities can then directly be used to build a Probability Mass Function (PMF): $f_K(k) = \Pr[k|\{(p_i, l_i)\}]$, with K the discrete random variable corresponding to the unknown subkey byte. This PMF will be needed to perform the key enumeration in Section 4. By contrast, in the case of non-profiled attacks (e.g. DPA [14] or CPA [6]), the best subkey hypothesis is not chosen based on probabilities, but on the value produced by a statistical distinguisher (namely, a difference-of-means test for Kocher's DPA and Pearson correlation coefficient for CPA). For these non-profiled attacks, there is thus no straightforward way to produce the PMF we need to enumerate the master keys. That is, the distinguisher outputs a ranking of the subkey candidates, which has no probabilistic meaning.

In order to apply our key enumeration algorithm, we need a way to extract probabilities from a non-profiled attack. For this purpose, we will use a natural extension of the non-profiled version of Schindler's stochastic approach [27]. Hence, we first recall how the non-profiled stochastic attack works [9]. A stochastic model $\theta(y)$ is a leakage model used to approximate the leakage function: $\mathsf{L}(y) \simeq \theta(y)$, where $\theta(y)$ is built from a linear basis $\mathsf{g}(y) = \{\mathsf{g}_0(y), ..., \mathsf{g}_{B-1}(y)\}$ chosen by the adversary (usually $\mathsf{g}_i(y)$ are polynomials in the bits of y). Evaluating $\theta(y)$ boils down to estimating the coefficients α_i such that the vector $\theta(y) = \sum_j \alpha_j \mathsf{g}_j(y)$ is a least-square approximation of the measured leakages l_i. The idea of a non-profiled stochastic attack is to build $|\mathcal{K}|$ stochastic models $\theta_k(y)$ by considering the data set $\{(p_i, l_i)\}$ under the assumption that the correct key hypothesis is k. These stochastic models are then used as a distinguisher: for a correct key hypothesis (and a relevant basis), the error between the predicted values and the actual leakage values should have a smaller standard deviation

[1] In order to lighten the notations, we omit the index $1 \leq i \leq q$ after the data sets.

than for a wrong key hypothesis. The pseudo-code of the attack is given in Algorithm 1. In general, an interesting feature of such attacks is that they allow trading robustness for precision in the models, by adapting the basis $\mathsf{g}(y)$. That is, a simpler model with less parameters is more robust, but a more complex model can potentially more accurately approximate the real leakage function.

Algorithm 1. Non-profiled stochastic attack

```
1: Acquire $\{(p_i, l_i)\}_{1 \leq i \leq q}$.
2: Choose a basis $\mathsf{g}(y)$.
3: for $k \in \mathcal{K}$ do
4:     Compute the S-box output hypotheses $y_{i,k} = \mathsf{S}(p_i \oplus k)$.
5:     Use the basis $\mathsf{g}(y)$, the data set and the subkey hypothesis $k$
       in order to build a stochastic model $\theta_k$.
6:     Compute the error vector $e_k$: $e_{i,k} = l_i - \theta_k(y_{i,k})$.
7:     Evaluate the precision of the model: $\sigma_k =$ standard deviation($e_k$).
8: end for
9: Choose $\hat{k} = \arg\min_k \sigma_k$.
```

3 Bayesian Extension of Non-profiled SCAs

As the straightforward application of a stochastic attack does not produce PMFs, we propose in this section to perform an additional Bayesian step after building the stochastic models. We show that this Bayesian model comparison produces probabilities, and that the criterion of maximizing the likelihood of the subkey is equivalent to minimizing the error vector standard deviation, meaning that this extension indeed ranks the subkeys in the same order as the standard non-profiled stochastic attack. As a bonus, we observe that this extension also gives us a very natural way to combine independent leakage samples in an attack. In the Bayesian version of the non-profiled stochastic attack, we perform a Bayesian hypothesis test on subkey candidates (under the assumption that the basis used for the stochastic attack is valid). It consists in estimating the probability of the observed data set assuming that they are produced from the model θ_k (i.e. $\Pr[\{(p_i, l_i)\}|\theta_k]$). Then, we use Bayes' theorem to deduce the likelihood of the models (and thus the probabilities of the subkey hypotheses) given the data (i.e. $\Pr[\theta_k|\{(p_i, l_i)\}]$), as described by the pseudo-code of Algorithm 2. A detailled derivation of relevant probabilities is given in Appendix A.

Algorithm 2. Bayesian non-profiled stochastic attack

```
1 to 8: Same as Algorithm 1.
9: Perform a Bayesian model comparison: evaluate for each subkey hypothesis the
likelihood $\Pr[\theta_k|\{(p_i, l_i)\}]$ using Bayes' theorem.
10: Choose $\hat{k} = \arg\max_k \Pr[\theta_k|\{(p_i, l_i)\}]$.
```

4 A New Algorithm for Combining Subkeys

Following the evaluation framework in [32], different metrics can be used to analyze the security of an implementation against side-channel attacks. Of particular interest in this work are the so-called "security metrics" (namely, the success rate and guessing entropy), of which the goal is to estimate the efficiency of a given distinguisher in exploiting the leakage to recover keys. Intuitively, a success rate of order o corresponds to the probability that the correct key is rated among the o first candidates provided by the distinguisher. The guessing entropy corresponds to the average number of keys to test before reaching the correct one. As suggested in [21], one can also consider a guessing entropy of order o, in order to capture the fact that in practice, only a maximum number of o keys can be tested by the evaluators. Empirical comparisons of distinguishers using such metrics have been performed in [31], but were limited to subkey recoveries (i.e. key bytes, typically). In the following, we consequently tackle the (most practically relevant) problem of how to efficiently estimate these metrics for master keys. In the extreme case (i.e. success rate of order 1), the solution is straightforward, as e.g. illustrated by the DPA contest [22]. We consider the general case of large lists and large orders, to carefully address the problem of the "conquer" part in a side-channel attack. The problem of extracting the rank of a correct key is equivalent to the problem of enumerating keys stated below.

Key-Enumeration Problem. The attacker obtains PMFs corresponding to d independent subkeys, each PMF taking n different values. The problem is to enumerate complete keys from the most probable to the least probable one.

In the following, we qualify an enumeration algorithm as *optimal* if it outputs key candidates in *nonincreasing* order of posterior probability. The term optimal refers to the fact that this order minimizes the expected number of key trials. Any non-optimal algorithm incurs an overhead in terms of trials during the key recovery phase. Besides, the more subjective term *efficient* relates to the computational and memory-related costs of the algorithm. For example, the naive algorithm for solving the enumeration problem generates the list of all possible key candidates, computes the corresponding likelihood values (by multiplying subkey probabilities) and sorts them accordingly. While optimal in the previously described sense, it can only be used with key candidates lists of limited size, and is therefore very inefficient. In the remainder of this section, we propose an algorithm that optimally solves the enumeration problem, and allows time and memory efficient key-enumeration, even when the number and size of subkey lists makes the naive approach untractable.

4.1 An Optimal and Efficient Key-Enumeration Algorithm

The key enumeration problem can be more readily understood as a geometric problem. We first consider the simpler bi-dimensional case (i.e. 2 subkeys). The key space can be identified with a compartimentalized square of length 1. The enumeration process is illustrated in Figure 1. The 4 columns (resp. rows) correspond to the four possible values of the first (resp. second) subkey, sorted by nonincreasing order of probability. Width and height correspond to the probability of the corresponding subkey. Let us denote by $k_i^{(j)}$ the j^{th} likeliest value for the i^{th} subkey. Then, the intersection of row j_1 and column j_2 is a rectangle corresponding to the key $(k_1^{(j_1)}, k_2^{(j_2)})$ with probability equal to the area of the rectangle. Using this geometric view, an optimal key enumeration algorithm outputs compartments by nonincreasing order of area. A solution to this problem is given in Algorithm 3 and corresponds to the following steps:

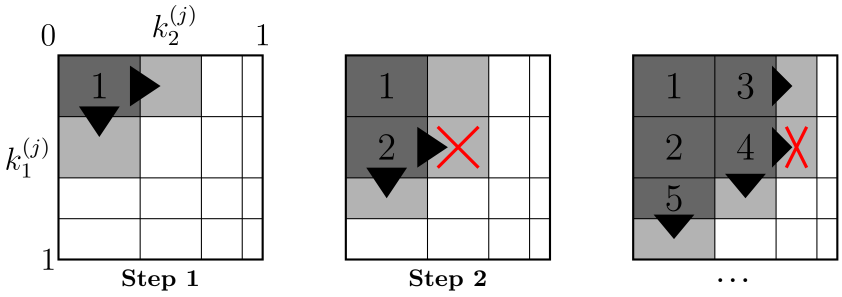

Fig. 1. Geometric representation of the proposed algorithm

Step 1. The most likely key is $(k_1^{(1)}, k_2^{(1)})$. Hence, we output this one first (represented in dark gray). At this point, the only possible next key candidates are the successors $(k_1^{(2)}, k_2^{(1)})$ and $(k_1^{(1)}, k_2^{(2)})$, shown in light gray. We denote by $\mathcal{F}$ this set of potential next candidates (where $\mathcal{F}$ is standing for frontier).

Step 2. Any new candidate has to belong to the frontier set. We extract the most likely candidate from this set and output it. It corresponds to rectangle 2 in our example. $\mathcal{F}$ is updated by inserting the potential successors of this candidate.

Next steps. Step 2 is repeated until the correct key is output, or if the size of the frontier set $\mathcal{F}$ exceeds the available memory space.

Note that in step 2, $(k_1^{(2)}, k_2^{(1)})$ is a more likely candidate than $(k_1^{(2)}, k_2^{(2)})$ by construction. Hence, $(k_1^{(2)}, k_2^{(2)})$ should not be inserted into $\mathcal{F}$ (this is represented on the figure by red crosses). There is a simple rule for handling such cases, which allows minimizing the memory requirements of our algorithm:

Rule 1. The set $\mathcal{F}$ may contain at most one element in each column and row.

Operations on the frontier set can be performed efficiently if candidate keys are stored in an ordered structure. Indeed, these operations simply consist in inserting new elements or finding the most likely element in the set and removing it. Using heaps, these manipulations are logarithmic in the size of the set. The test of Rule 1 can be implemented using arrays of Boolean values.

Algorithm 3. Optimal key-enumeration.

```
$\mathcal{F} \longleftarrow \{(k_1^{(1)}, k_2^{(1)})\}$;
**while** $\mathcal{F} \neq \emptyset$ **do**
    $(k_1^{(i)}, k_2^{(j)}) \longleftarrow$ most likely candidate in $\mathcal{F}$;
    Output $(k_1^{(i)}, k_2^{(j)})$;
    $\mathcal{F} \longleftarrow \mathcal{F} \setminus \{(k_1^{(i)}, k_2^{(j)})\}$;
    **if** $i + 1 \leq \#k_1$ and no candidate in row $i + 1$ **then**
        $\mathcal{F} \longleftarrow \mathcal{F} \cup \{(k_1^{(i+1)}, k_2^{(j)})\}$;
    **end if**
    **if** $j + 1 \leq \#k_2$ and no candidate in column $j + 1$ **then**
        $\mathcal{F} \longleftarrow \mathcal{F} \cup \{(k_1^{(i)}, k_2^{(j+1)})\}$;
    **end if**
**end while**
```

Generalization to Multiple Lists. In practice, one often has to merge together more than two lists of subkeys. Straightforward extensions of our algorithm to higher dimensions lead to either suboptimal or slow rules for frontier set reduction. On the one hand, the direct transposition of Rule 1 will minimize memory, but implies adjacency tests in multiple dimensions, leading to unacceptable reductions of the enumeration speed. On the other hand, simplifying the rule in order to maintain a good enumeration speed implies the storage of many non-necessary candidates in the frontier set, which rapidly leads to unsustainable memory requirements. As a result, and in order to obtain good results for more than two lists, we apply a recursive decomposition of the problem.

For this purpose, we only use the algorithm for merging two lists, and its outputs are combined to form larger subkey lists which are in turn merged together. This way, merging n lists is done by merging two lists $n - 1$ times. The order of merging is such that lists merged together are of similar sizes. Taking the example of the AES, we notice that enumerating 128-bit keys is done by merging two lists of size 2^{64}. Such lists cannot be generated or stored efficiently. Fortunately, we can instead generate these lists only as far as required by the key enumeration. Whenever a new subkey is inserted in the candidate set, we get it from the enumeration algorithm applied to the lower level (e.g. 64-bit subkeys are obtained by merging two 2^{32} element lists), and so on. This ensures that the storage and enumeration effort are minimized. The process is illustrated in Figure 2. Enumerating 16-byte keys consists in enumerating subkeys taken from the two 2^{64}-element lists $k_{0,\ldots,7}^{(j)}$ and $k_{8,\ldots,15}^{(j)}$, which in turn are built using four 2^{32}-element lists $k_{0,\ldots,3}^{(j)}$, $k_{4,\ldots,7}^{(j)}$, etc. This process is repeated until we reach the original 2^8-element subkey distributions. The recursive decomposition combined with our *lazy evaluation* technique keep computations and memory requirements to a minimum and allow us to enumerate a large number of key candidates.

The investigations in this paper have connections with previous works in statistical cryptanalysis. We provide a detailled review in Appendix B.

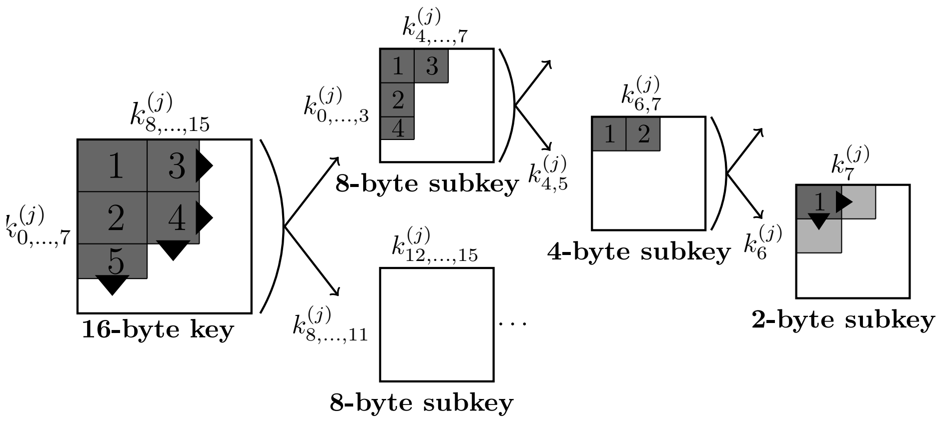

Fig. 2. Recursive enumeration from multiple lists of key candidates

5 Experiments

In order to validate our approach, we led several experiments. The cipher under investigation was the AES, with key size of 128 bits. Our side-channel attacks targeted the output of the S-boxes in the first round, resulting in 16 independent subkey probability mass functions. We used the same assumptions as in Section 2 and considered simulated leakages, following a Hamming weight leakage model on the S-box output, with an independent additive Gaussian noise, i.e. $\mathsf{f}(y) = \mathrm{HW}(y) + \mathcal{N}(0, 4^2)$. Note that the type of experiments performed (i.e. analyzing the impact of key enumeration) is essentially independent of both the cipher and experimental setup. We carried out both template attacks with perfect profiling (since the leakage function is known) and non-profiled Bayesian stochastic attacks assuming a linear basis made of the S-box output bits. The enumeration was performed using our open-source optimized implementation [1].

5.1 Comparing Optimal and Probabilistic Key-Enumerations

Table 1 gives some performance results for key enumeration obtained on our setup (Intel core i7 920 running a 64-bit Ubuntu 11.04 distribution). These comparative results show that both the sampling algorithm and ours can output key candidates at essentially the same speed. The results for the enumeration algorithm described in [21] are also given. As expected, they exhibit larger memory requirements, which bounds the number of key candidates that can be enumerated and increases time. In practice, this method is limited to 2^{20} candidates. By contrast, our algorithm allows enumerating 2^{32} keys using less than 1GB.

Next, as mentioned in Section 4, Algorithm 3 performs key trials in the best possible order, therefore minimizing the enumeration effort at the cost of a growing amount of memory space. By contrast, the probability-driven algorithm may miss some key candidates and output some more than once. In order to illustrate

Table 1. Practical comparison of key-enumeration algorithms (time, min-max memory)

#trials	2^{16}	2^{20}	2^{24}	2^{28}	2^{32}	2^{33}	2^{35}	2^{37}
Sampling	0.04s	0.31s	10.1s	160s	2560s	X	X	X
[21]	0.96s 118MB	18.1s 7.7GB	X	X	X	X	X	X
Ours	0.01s 100KB 500KB	0.25s 2MB 3MB	5s 3.9MB 12MB	100s 30MB 80MB	1700s 140MB 530MB	4058s 190MB 540MB	5.4h 560MB 1.1GB+2.3HDD	28.2h 0.9GB+1.7HDD 1.1GB +6.1HDD

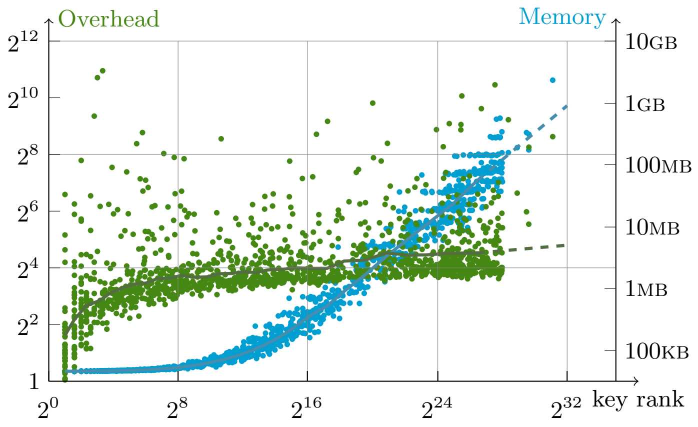

Fig. 3. Overhead of the probability-driven method in function of the key rank (green), and memory requirements of the deterministic enumeration (light blue)

these differences we led the following experiment. A large number of side-channel attacks followed by a key recovery were performed, and we measured the key rank (which is also the number of trials for the optimal algorithm), the expected number of trials for the probability-driven algorithm and the memory used during optimal enumeration. Figure 3 illustrates the expected overhead of the probability-driven method over the deterministic one in terms of key trials (green dots, left scale) and the memory cost of the enumeration algorithm (blue dots, right scale). We observe that the probability-driven algorithm requires more key trials on average to complete an attack. The overhead increases consistently, and the median of the expected ratio value (green curve) appears to tend towards a linear relation on the log-log scale. An approximated power law gives 16 for 2^{16}, 21 for 2^{32}, an extrapolated 36 for 2^{40}. In some cases, we also observe overheads very far from the median value (well over 1000), even when the correct key is ranked among the 4 first ones. On top of this *expected* number of trials, we have to consider that, since the probabilistic method follows a geometric law, the number of key trials will have a very large variance (approximately the square of expected value). This makes the probabilistic method both more costly and less reliable than our deterministic algorithm. Besides, the memory space requirement of the enumeration method also appears to follow a power law. Enumerating up to 2^{32} requires only 1GB of memory. Extrapolations predict a cost of 70GB for 2^{40} keys.

5.2 Application of Key-Enumeration to Side-Channel Attacks

Figure 4 illustrates the success rate of different orders for a template attack, in function of the number of traces measured. The alternated light and gray zones correspond to the evolution of the success rate each time the number of tested keys is multiplied by 16. The rightmost dark gray curve is obtained by only testing the first key candidate, the first light gray curve by testing 2^4 keys, then 2^8, ... We again considered the optimal and the probabilistic algorithms, for different number of enumerated key candidates. The optimal enumeration was led up to 2^{32} candidates, and the probabilistic one up to 2^{28}.

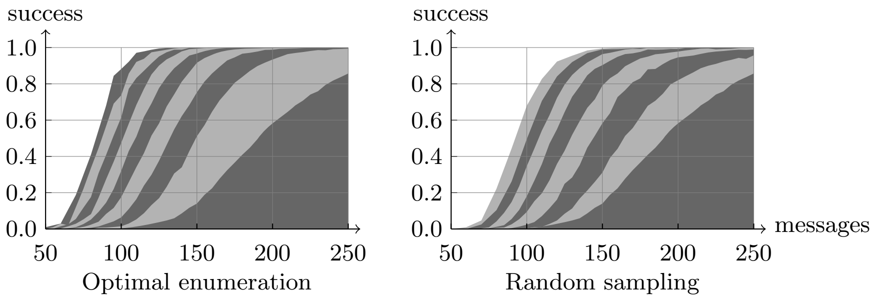

Fig. 4. Success rate of template attacks. Left: enumeration, right: sampling.

As expected, allowing more key candidates to be tested can dramatically increase the efficiency of a key recovery. For 120 messages measured, the best key candidate is the correct one about 2% of the time, while there is a 91% chance for the correct key to be found among the first 2^{24} candidates with the optimal enumeration algorithm (or an 84% chance with the probability driven method). In other words, increasing the number of key trials significantly improves the success rate, thereby providing a tradeoff between the data and time complexities of the attacks. As in the previous subsection, we also observe that the optimal enumeration algorithm leads to higher success rates compared to the random sampling for a given number of key trials, at the cost of additional memory requirements.

Figure 5 provides an orthogonal view of the problem: for a given number of traces, one can increase the key success rate by enumerating more key candidates. The figure shows the cumulative probability function (CDF) of key recoveries for an attack with a fixed number of traces, in function of the number of key trials. The PMFs used for this experiment are output by two template attacks. The first attack (left) targets only 2 key bytes, the second (right) targets all 16 key

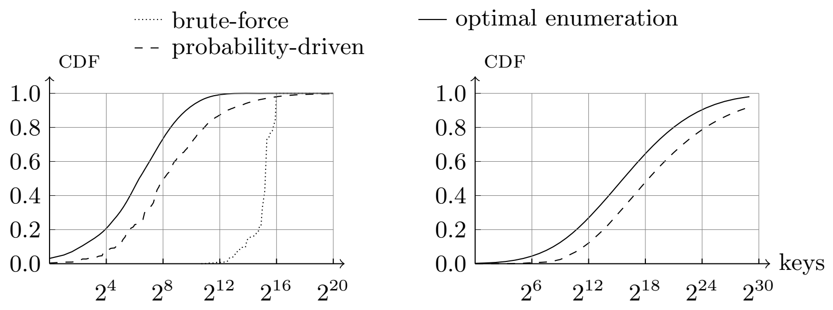

Fig. 5. Enumeration success rates. Left: 2 S-boxes, right: 16 S-boxes.

bytes. As expected, the cumulative probability starts from 0 and reaches 1 once a sufficient number of keys have been tested. Also, the brute force testing is only possible in the left case (i.e. when we can enumerate the full list).

In general, the side-channel information allows obtaining high success rates with a limited number of key trials (here, up to 2^{30}). More importantly, the figure again confirms the interest of the deterministic algorithm in terms of "number of keys to test". Reaching a similar success rate with the probabilistic algorithm requires between 2^2 and 2^4 more tests, depending on the success rates.

6 Conclusion

This paper complements standard side-channel cryptanalysis by investigating the improvements obtained by adversaries with non-negligible computational power. We first proposed an extension of non-profiled stochastic attacks that outputs probability mass functions, providing us with the likelihood values of subkey candidates. Next, we proposed a new and deterministic key enumeration algorithm, in order to take advantage of these likelihood values. Experiments show that this order-optimal enumeration algorithm is more efficient than a sampling-based algorithm from Eurocrypt 1991. In particular, the probabilistic algorithm suffers from its underlying geometric law, that implies an increasingly large overhead over the deterministic method (in terms of key trials), as the number of keys to enumerate increases. The deterministic method additionally allows removing the possibility of worst cases, which makes it a particularly suitable solution for side-channel security evaluations. Finally, our proposal significantly reduces the memory requirements of a deterministic algorithm from SAC 2010, making it the best practical solution for enumeration of up to 2^{40} keys.

As a result, the solutions in this paper allow us to properly trade side-channel measurements for offline computations. They create a bridge between classical DPA and brute-force key recovery, where information extracted through side-channels is used to improve an exhaustive search. Hence, an interesting research problem is to compare computationally-enhanced DPA attacks with other types of more computational side-channel attacks, e.g. based on the detection of collisions. The application of enumeration in statistical cryptanalysis is another possible direction for further investigations. Note finally that the complete key recoveries we considered in this work can possibly be performed in a ciphertext-only context. That is, the adversary can evaluate a key candidate by partially decrypting the ciphertext and computing the probability of previous-round leakages, which will only be non-negligible when decrypting with the correct key.

Acknowledgements. This work has been funded by the B-CCENTRE with the financial support from the Prevention of and Fight against Crime Programme of the European Union, by the the ERC project 280141 (acronym CRASH), by the Walloon regions MIPSs and SCEPTIC projects. F.-X. Standaert is a Research Associate of the Belgian Fund for Scientific Research (FNRS-F.R.S).

References

1. `http://perso.uclouvain.be/fstandae/source_codes/enumeration/`
2. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
3. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
4. Bogdanov, A.: Improved Side-Channel Collision Attacks on AES. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 84–95. Springer, Heidelberg (2007)
5. Bogdanov, A., Kizhvatov, I., Pyshkin, A.: Algebraic Methods in Side-Channel Collision Attacks and Practical Collision Detection. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 251–265. Springer, Heidelberg (2008)
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, Quisquater (eds.) [11], pp. 16–29
7. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
8. Dichtl, M.: A new method of black box power analysis and a fast algorithm for optimal key search. J. Cryptographic Engineering 1(4), 255–264 (2011)
9. Doget, J., Prouff, E., Rivain, M., Standaert, F.-X.: Univariate side channel attacks and leakage modeling. J. Cryptographic Engineering 1(2), 123–144 (2011)
10. Johansson, T. (ed.): FSE 2003. LNCS, vol. 2887. Springer, Heidelberg (2003)
11. Joye, M., Quisquater, J.-J. (eds.): CHES 2004. LNCS, vol. 3156. Springer, Heidelberg (2004)
12. Junod, P.: On the Optimality of Linear, Differential, and Sequential Distinguishers. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 17–32. Springer, Heidelberg (2003)
13. Junod, P., Vaudenay, S.: Optimal key ranking procedures in a statistical cryptanalysis. In: Johansson (ed.) [10], pp. 235–246
14. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

15. Ledig, H., Muller, F., Valette, F.: Enhancing collision attacks. In: Joye, Quisquater (eds.) [11], pp. 176–190
16. Lenstra, A.K., Verheul, E.R.: Selecting cryptographic key sizes. J. Cryptology 14(4), 255–293 (2001)
17. Mangard, S., Oswald, E., Standaert, F.-X.: One for all – all for one: unifying standard differential power analysis attacks. IET Information Security 5(2), 100–110 (2011)
18. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
19. Meier, W., Staffelbach, O.: Analysis of Pseudo Random Sequences Generated by Cellular Automata. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 186–199. Springer, Heidelberg (1991)
20. Oren, Y., Kirschbaum, M., Popp, T., Wool, A.: Algebraic Side-Channel Analysis in the Presence of Errors. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 428–442. Springer, Heidelberg (2010)
21. Pan, J., van Woudenberg, J.G.J., den Hartog, J.I., Witteman, M.F.: Improving DPA by Peak Distribution Analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 241–261. Springer, Heidelberg (2011)
22. T. ParisTech. DPA contest v2, `http://www.dpacontest.org/v2/index.php`
23. C. K. L. Recommendation, `http://www.keylength.com/`
24. Renauld, M., Standaert, F.-X.: Algebraic Side-Channel Attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010)
25. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009)
26. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 109–128. Springer, Heidelberg (2011)
27. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
28. Schramm, K., Leander, G., Felke, P., Paar, C.: A collision-attack on AES: Combining side channel- and differential-attack. In: Joye, Quisquater (eds.) [11], pp. 163–175
29. Schramm, K., Wollinger, T.J., Paar, C.: A new class of collision attacks and its application to DES. In: Johansson (ed.) [10], pp. 206–222
30. Seshadri, N., Sundberg, C.-E.W.: List viterbi decoding algorithms with applications. IEEE Transactions on Communications 42(2/3/4), 313–323 (1994)
31. Standaert, F.-X., Gierlichs, B., Verbauwhede, I.: Partition *vs.* Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 253–267. Springer, Heidelberg (2009)
32. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)

A Bayesian Extension of Non-profiled SCAs

We now show how to compute these probabilities, starting with the probability of observing the data set $\{(p_i, l_i)\}$ assuming it is produced by the model θ_k, using the subkey hypothesis k. This probability is computed by multiplying the probabilities of each individual event (p_i, l_i) of the data set:

$$\begin{aligned}\Pr[\{(p_i, l_i)\}|\theta_k] &= \Pr[\{(p_i, l_i)\}|\theta_k, K = k],\\ &= \prod_{i=1}^{q} \mathcal{N}(l_i, \theta_k(\mathsf{S}(p_i \oplus k)), \sigma_k),\end{aligned}$$

where $\mathcal{N}(x, \mu, \sigma)$ is the value of the normal distribution of mean μ and standard deviation σ evaluated at point x. If we denote the S-box output hypotheses as $y_{i,k} = \mathsf{S}(p_i \oplus k)$, the previous equation can be rewritten as:

$$\Pr[\{(p_i, l_i)\}|\theta_k] = \prod_{i=1}^{q} \frac{1}{\sqrt{2\pi}\ \sigma_k} \exp^{-\frac{1}{2\sigma_k^2}(l_i - \theta_k(y_{i,k}))^2}.$$

Since $\sigma_k^2 = \sum_{i=1}^{i=q}(l_i - \theta_k(y_{i,k}))^2/q$ (see Algorithm 1), if we use all q measurements, we can simplify the exponent and move all constants coefficients that do not depend on k in a normalization term Z, that is:

$$\Pr[\{(p_i, l_i)\}|\theta_k] = Z\sigma_k^{-q}. \tag{1}$$

Then, using Bayes' theorem, we deduce the probabilities of the subkeys from the respective likelihood values of the models θ_k given the data (in other words, we perform a Bayesian model comparison):

$$\begin{aligned}\Pr[k|\{(p_i, l_i)\}] &= \Pr[\theta_k|\{(p_i, l_i)\}],\\ &= \frac{\Pr[\{(p_i, l_i)\}|\theta_k].\Pr[\theta_k]}{\Pr[\{(p_i, l_i)\}]}, \quad \text{(Bayes' theorem)}\\ &= \frac{\Pr[\{(p_i, l_i)\}|\theta_k].\Pr[\theta_k]}{\sum_{k' \in \mathcal{K}} \Pr[\{(p_i, l_i)\}|\theta_{k'}].\Pr[\theta_{k'}]}.\end{aligned}$$

Assuming a uniform prior $\Pr[\theta_k]=\Pr[k]=\frac{1}{|\mathcal{K}|}$ and using Equation 1, we get:

$$\Pr[k|\{p_i, l_i\}] = \frac{\sigma_k^{-q}}{\sum_{k'} \sigma_{k'}^{-q}}.$$

From these probabilities, we can directly build the PMF required for key enumeration. Note that these likelihood values are not exactly the same as the ones we used in template attacks. In the last case, the characterization of a device allows exploiting a precise estimation of the leakage distributions. By contrast, in a Bayesian non-profiled stochastic attack, they depend on the basis $\mathsf{g}(y)$ chosen by the adversary. Finally, the subkey hypotheses can be ranked according to the likelihood values of the corresponding model given the data, that is:

$$\begin{aligned}\hat{k} &= \arg\max_k \sigma_k^{-q}, \\ &= \arg\min_k \sigma_k,\end{aligned}$$

i.e. providing the same ranking as for the original non-profiled stochastic attack. Besides their use for key enumeration in the next section, an appealing property of using probabilities instead of other criteria (e.g. like a correlation coefficient) is that combining independent measurement points becomes very natural. Let us suppose that our implementation leaks information at two different times: $\mathsf{L}_{t_0} = \mathsf{f}_{t_0}(x) + n_{t_0}$ and $\mathsf{L}_{t_1} = \mathsf{f}_{t_1}(y) + n_{t_1}$ (with n_{t_i} a Gaussian noise). The Bayesian writing makes it straightforward to combine their corresponding probabilities, by multiplication and normalization. Note also that the proposed attack can additionally be seen as a generalization of DPA or CPA, by simply replacing the leakage basis by a single-bit or Hamming weight model (as observed in [17]).

B Comparison with Previous Works

The investigations in this paper have strong connections with previous works in the area of statistical cryptanalysis. In particular, the problem of merging *two* lists of subkey candidates was encountered by Junod and Vaudenay [13]. The small cardinality of the lists (2^{13}) was such that the simple approach that consists in merging and sorting the lists of subkeys was tractable. Dichtl considered a similar problem of enumerating key candidates by decreasing order of probabilities, thanks to partial information obtained for each key bit individually [8]. We tackle the more general and challenging problem of exploiting any partial information on subkeys. A frequent reference for solving this problem, i.e. enumerating many keys from lists that cannot be merged, is the probabilistic algorithm that was proposed in [19]. In this work, the attacker had no access to the subkey distributions but was able to generate subkeys according to them. Hence, the solution proposed was to enumerate keys by randomly drawing subkeys according to these distributions. Implementing this algorithm is equivalent to uniformly picking up a point in the square of Figure 1 and testing the corresponding key. This does not require any memory but the most probable keys may be drawn many times, leading to useless repetitions. Indeed given a correct key with probability p the number of keys to try before it is found follows a geometric distribution with parameter p and thus has an expected value equal to $1/p$ with a variance of $\frac{1-p}{p^2}$. By contrast, for Algorithm 3, this number of keys to test is *at most* $\lfloor 1/p \rfloor$ (usually much less). Actually, our algorithm will output exactly n keys before the correct one if it is ranked in the n-th position, removing the variance issues of the probabilistic test. Also in practice, the probability-driven approach tends to lead to much more tests than optimal deterministic enumeration, as will be illustrated experimentally in the next section.

Next, and in terms of complexities, it is easy to see that the probability-driven algorithm can output new keys in constant time and has a very small memory requirement. The case of our deterministic enumeration algorithm is more difficult. The use of heaps for the frontier set and the recursive decomposition of the problem point towards a logarithmic time complexity. However, it appears from the experiments in the next section that the algorithm enumerates keys in amortized time close to constant. Summarizing, both methods lead to a linear time complexity in the total number of key candidates that are output, with the enumeration algorithm also requiring a sub-linear amount of memory.

As previously mentioned, an enumeration algorithm similar to ours was proposed in a paper by Pan, van Woudenberg, den Hartog and Witteman [21]. It also enumerates key candidates in optimal order, but the reduction rule 1 is not used, nor the recursive decomposition that allows us to efficiently apply rule 1 with more than two lists. Therefore, the frontier set of their algorithm is not reduced, and the memory requirements are much larger. In practice, since the main limitation for optimal key enumeration appears to be memory, this non-minimal version of enumeration does not allow an adversary to output a large number of key candidates. Moreover, handling a larger frontier set implies time complexity penalties, which makes our new algorithm faster than this previous one. Implementation results confirming these claims are given in the next section.

Finally, we note that another related problem is list decoding of convolutional codes through the Viterbi algorithm (see, e.g. [30]). However, and as previously mentioned, enumeration and decoding are not the same problems and the latter one only makes sense in the presence of redundancy, which does not exist when subkeys are independent of one another. An attempt at using such an algorithm would either lead to combinatorial explosion (i.e. 2^{128} possible states) for a deterministic version, or require very large amounts of memory when using approximate solutions such as beam search. Moreover, list-decoding algorithms are generally designed to output a small number of most likely candidates determined *a priori*, whereas we typically target the enumeration of 2^{32} or more master key candidates, continuing enumeration until the correct key is found.