IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 277–300. DOI:10.46586/tches.v2023.i1.277-300

MCRank: Monte Carlo Key Rank Estimation for Side-Channel Security Evaluations

Giovanni Camurati[1], Matteo Dell'Amico[2] and François-Xavier Standaert[3]

[1] ETH Zurich, Zurich, Switzerland, giovanni.camurati@inf.ethz.ch
[2] University of Genoa, Genoa, Italy, matteo.dellamico@unige.it
[3] UC Louvain, Louvain, Belgium, fxstandae@uclouvain.be

Abstract. Key rank estimation provides a measure of the effort that the attacker has to spend bruteforcing the key of a cryptographic algorithm, after having gained some information from a side channel attack. We present `MCRank`, a novel method for key rank estimation based on Monte Carlo sampling. `MCRank` provides an unbiased estimate of the rank and a confidence interval. Its bounds rapidly become tight for increasing sample size, with a corresponding linear increase of the execution time. When applied to evaluate an AES-128 implementation, `MCRank` can be orders of magnitude faster than the state-of-the-art histogram-based enumeration method for comparable bound tightness. It also scales better than previous work for large keys, up to 2048 bytes. Besides its conceptual simplicity and efficiency, `MCRank` can assess for the first time the security of large keys even if the probability distributions given the side channel leakage are not independent between subkeys, which occurs, for example, when evaluating the leakage security of an AES-256 implementation.

Keywords: Key rank estimation · Side channel attacks · Monte Carlo methods

1 Introduction

Side-channel attacks leverage some physical leakage (e.g., power consumption [KJJ99], electromagnetic emissions [AARR03], acoustic emissions [GST14]) to recover information about secrets like cryptographic keys manipulated by concrete devices. Such attacks can be non-profiled such as Brier et al.'s Correlation Power Analysis (CPA) [BCO04] or profiled such as Chari et al.'s template attack [CRR03]. These attacks are divide-and-conquer, meaning that they return the probabilities of some enumerable parts of a master key (sometimes call subkeys) one by one. For example, in the AES-128 case, one typically gathers information byte per byte. In the best case for the attacker, sorting the probability lists and selecting the highest value for each byte returns the correct key. However, if the attack was not completely successful, the right key has lower probability and is not selected. To recover it, the attacker then has to enumerate keys by decreasing probability, until the right one is found [VGRS13]. Quite naturally, such an enumeration work is limited by the computational power of the evaluator and it can be prohibitive as the attack's complexity increases, up to the point where it becomes the bottleneck of a security evaluation. As a result, key rank estimation algorithms have been introduced [VGS13]. The key rank is defined as the number of keys with higher probability than the right key (already known by the evaluator), and it provides an interesting measure of the computation effort the attacker should spend to fully recover the key. Key rank estimation is an active research field with many algorithms proposed [BLv15, GGP+15, MOOS15, MMOS16, CP17, Gro18, DW19c, DW19b, DW21].

Key rank algorithms generally compete in terms of execution time, tightness of the bounds provided, and memory requirements. Most of the published algorithms are both efficient

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

and accurate enough to deal with standard symmetric cryptography key sizes (e.g., 128-bit keys), and they return an upper and lower bound on the rank. The histogram enumeration approach proposed by Glowacz et al. [GGP+15] is a popular example due to its simplicity, and it can be extended to scale well for very large keys (e.g., key-load attack on RSA) [Gro18].

Interestingly, the problem of key rank estimation shares several similarities with the problem of password strength evaluation in the field of system security. In short, a password is strong with respect to a probabilistic model for password generation if the model generates many other passwords with higher probability than the right one. Hence the similarity between key rank estimation (i.e., estimating how many keys have higher probability than the right one after a side channel attack) and password strength evaluation (i.e., estimating how many passwords are generated with higher probability than the right one by a model for password generation). By splitting the password in multiple dimensions (similar to subkeys in side channel attacks), a recent work by David and Wool proposed to estimate password strength using a key rank estimation approach borrowed from the literature on side channels [DW19a]. In this paper, we take the opposite approach and ask ourselves whether it is possible to improve the state of the art for side channels by leveraging the advantages of algorithms proposed for password strength estimation.

We answer this question positively by proposing a novel method for key rank estimation based on Monte Carlo sampling, denoted as `MCRank`, which is inspired by literature on password strength [DF15], but which also take into account some specific challenges in side channel analysis. In particular, `MCRank` accepts both scores and probabilities as input, and it uses a rescaling algorithm to adjust the sampling probability and achieve high accuracy even if the input scores/probabilities are unbalanced. We show that `MCRank` provides an unbiased estimate of the rank, and we compute probabilistic bounds for it (e.g., a $\pm 3\sigma$ interval corresponding to 99.7 % confidence for a Gaussian distribution). We also show that `MCRank` quickly achieves good tightness (e.g., less than 1 bit) by tuning the sample size n, and can be orders of magnitude faster than the state of the art histogram method [GGP+15] for comparable tightness (with an execution time growing linearly with the sample size). `MCRank` is evaluated with both template and profiled correlation attacks on simulated traces, and deep-learning attacks on real traces from a protected implementation from the well known ASCADv2 dataset. In addition, `MCRank` scales better than previous solutions that optimize for large keys [CP17, Gro18], achieving high accuracy in low execution time for key-load attack on RSA with keys up to 2048 bytes. Despite other state of the art solutions are admittedly sufficient to deal with most practical cases (in particular for symmetric cryptography), this first part of the paper provides a handy complement to previous work, with a rank estimation algorithm that combines conceptual simplicity and excellent performances (even for very large keys that could not be dealt with efficiently before).

More interestingly, the second contribution of the paper deals with the analysis of rank estimation even when the probability distributions (given the side channel leakage) are non-independent among subkeys. This happens, for example, when evaluating an AES-256 implementation. While solutions were proposed in specific contexts, such as key enumeration for attacks on elliptic curves [LvVW14] and collision attacks [WLW17], we are not aware of a generic solution for symmetric key algorithms like AES-256. In this case, the second part of the key is used in a second round so that a successful attack requires attacking the first round and then using the result to attack the second round. For each possible key explored in the first round, the attack on the second round should be repeated to find the probabilities for the second part of the key (more details in Section 2). We show that `MCRank` can be used to efficiently evaluate such contexts.

We note that one reason for the complexity of key rank estimation algorithms is requiring deterministic bounds. So the reason why our approach improves performances primarily comes from the probabilistic nature of `MCRank`. While such a probabilistic result can some-

times be a drawback, we believe that probabilistic rank estimation can be just as useful in practice, when combined with appropriate confidence intervals. We note also that some existing algorithms leverage statistical sampling approaches as a post-processing step to tighten the bounds found by the classical approach [BLv15].

2 Background and Related Work

Side Channel Attacks When a cryptographic algorithm is implemented in software or hardware, its execution might produce data-dependent physical emissions that can reveal information about the secret key. Examples side channel leakage are power consumption [KJJ99], and electromagnetic [AARR03] or acoustic [GST14] emissions.

Template Attacks on AES-128 In this paper we focus on template attacks [CRR03] on the Advanced Encryption Standard (AES) [Pub01]. Let p_i and k_i be a byte of the plaintext $p=(p_0,...,p_{15})$ and of the key $k=(k_0,...,k_{15})$, respectively. We denote the secret key as $k^*=(k_0^*,...,k_{15}^*)$, to distinguish it from a generic key k. At the first round of AES-128, p_i and k_i^* are xored and passed to a substitution box S_{box} (i.e., $y_i=S_{box}[p_i \oplus k_i^*]$). The goal of the attacker is to recover the value of each byte k_i^* of the secret key, given the leaking variable $y_i=S_{box}[p_i \oplus k_i^*]$ and many measurements (physical observations) l of the corresponding physical leakage.

In a first step, the attacker characterizes (a copy of) the target device by measuring the leakage for a large number of known random p_i and k_i. For each possible value y of the leaking variable Y, the attacker estimates the mean and covariance of the physical leakage L. Under the further assumption of Gaussian distribution, the attacker computes the conditional probability density function of L given Y (i.e., $pdf(L=l|Y=y_i)$).

In a second step, the attacker collects a leakage trace l_0 on the target device for known p_i and unknown (secret) k_i^*. Using the conditional probability density function computed before, the attacker can assign a probability to each possible guess k_i^g of the value of k_i, by computing $pdf(L=l|Y=y_i^g)$ (with $y_i^g=S_{box}[p_i \oplus k_i^g]$). The result can be improved using multiple attack traces l_j (corresponding to the same secret k_i^*) and computing the maximum likelihood $d(k_i^g)=\prod_j pdf(L=l_j|Y=y_i^g)$. By collecting more and more traces, the attacker will gather information that will likely increase the probability of k_i^* and therefore lower its rank among the possible values of k_i.

To recover the full AES-128 key, the attacker repeats 16 independent template attacks on the first round (one for each byte p_i and k_i^* of the plaintext and key). Finally, the attacker performs a key enumeration [VGRS13] to find the key even when the side channel attack has not fully recovered all bits, as explained in Section 1.

For software implementations, it is common to assume that the leakage is proportional to the Hamming Weight of y_i (i.e., leakage function $HW[y_i]$) with the addition of Gaussian noise. This assumption is commonly used for simulations. On the attack side, using $HW[y_i]$ allows building only 9 templates for all possible values of $HW[y_i]$ instead of 256 templates for all possible values of y_i.

Extension to AES-256 In AES-256, the first 128 bits of the key can be recovered by applying the same template attack on the first round as for AES-128. However, the second 128 bits of the key are xored with the output of the first round that is unknown to the attacker (it depends on the secret key). The simplest extension of the template attack to the AES-256 case consists in first attacking the first round to recover the first half of the key, and then using the result to attack the second round and recover the second half of the key. Clearly, the success of the second step depends on the success of the first. Consequently, enumeration becomes more complex. The attacker tries D_1 values of the first part of the key. For each of these values, the attacker has to repeat the attack on the second round, and enumerate D_2 values of the second part of the key. This requires a total of $1+D_1$ template attacks and

it leads to a guessing space of $D_1 \cdot D_2$ values. This is an interesting example of side channel attack in which the key is recovered in multiple steps that are not independent. Improvements in the case of known plaintext and ciphertext have been shown by Wurcker [Wur19].

Key-load attack on RSA Template attacks are also useful to recover the key or the plaintext from the leakage produced when it is loaded in memory. In this case, each independent byte k_i of the key is attacked directly (i.e., $y^g = k_i$). In this case, it is better to use the identity leakage $y^g = k_i$ instead of the Hamming Weight model, since using the Hamming Weight directly on k means that it is impossible to distinguish key bytes k_i that have the same Hamming Weight. Key-load attacks on large RSA keys are a common scenario targeted by key enumeration algorithms [CP17, Gro18].

Profiled correlation attacks Profiled correlation attacks [DS16] are a simple extension of correlation attacks. During the profiling step, the attacker estimates the average value of the leakage for each possible value of y_i. This serves as a model $m(y_i)$ of the leakage. During the attack phase, the attacker uses Pearson's correlation coefficient between leakage and model $d(k_i^g) = \rho(m(y_i^g), l)$ to assign a score to each possible value k_i^g. Ideally, with enough attack traces the real value k_i^* emerges as the one with the highest correlation score. The profiling step does not necessarily require a set of profiling traces, it can also be performed on a subset of the attack traces. Profiled correlation attacks are different from templates because (i) they capture only a first order relationship between leakage and model, and (ii) they return correlation scores instead of probabilities. However, scores can be transformed in probabilities following the Bayesian extension proposed in [CPS16].

Deep-Learning attacks and the ASCADv2 dataset In recent years, side channel attacks based on Deep-Learning have gained traction. In essence, they formulate the problem of profiled side channel attacks (e.g., template attacks) as a classification problem that can be solved with a deep neural network [BPS+20]. For example, during learning the deep neural network is trained to find an approximation $g(l, y)$ of the conditional probability $P(Y = y | L = l)$. During attack, each byte value is assigned a maximum likelihood score $d(k_i^g) = \prod_j g(Y = y_i^g, L = l_j)$. Ideally, with enough attack traces the real k_i^* emerges as the one with highest score. To facilitate reproducible research on deep learning attacks, ASCAD [MS21] has been proposed as a reference benchmark consisting of a protected implementation of AES-128, datasets of pre-recorded traces, and reference attacks. In this paper, we use its improved version ASCADv2 [MS21], which proposes two attacks against an implementation of AES-128 protected by affine masking and shuffling.

Key rank Let us assume that the attacker has learned the probabilities $p(k_i)$ for each possible value of k_i and each sub-key using a side channel attack. Assuming independent events, the probability of a full key $k = (k_0, k_1, ..., k_{m-1})$ is the product of the sub-key probabilities $p(k) = \prod_{i=0}^{m-1} p(k_i)$ and the rank of the real key is defined as $R_{k^*} = |\{k' | p(k') > p(k^*)\}|$. These are reasonable assumptions, for example, when attacking AES-128 with an independent template attack for each subkey. Correlation attacks return scores $d(k_i)$ which are not directly probabilities. The rank of a full key can still be defined as $d(k) = \prod_{i=0}^{m-1} d(k_i)$, and the rank becomes $R_{k^*} = |\{k' | d(k') > d(k^*)\}|$. However, enumeration strategies based on scores may not be optimal [CPS16]. The key rank defined on scores is a good measure of the attempts that the attacker has to make to find the key based on the specific attack that returned the scores, but is not necessarily the smaller one, as an attacker working with probabilities might find a better result. In practice, key rank estimation algorithms such as [GGP+15] work well with both probabilities and scores. While it is recommended to improve the attack so that it returns balanced probabilities (for example, using a simple Bayesian extension of a correlation attack [CPS16]), we focus on computing R_{k^*} from the

output of an attack (be it scores or probabilities), leaving aside the orthogonal problem of finding the optimal attack strategy. We will also devise a strategy to estimate the rank for AES-256, where the attacks on first and second half of the key are not independent.

State-of-the-art key rank estimation algorithms Over the last decade, several key rank estimation algorithms have been proposed. In the seminal work of Veyrat et al. [VGS13], the key space is organized as a multi-dimensional volume in which keys with higher/lower probability than the real key are separated by a convex surface. Upper and lower bounds on the key rank can be estimated by iteratively carving sub-volumes from both sides.

The Histogram Enumeration method [GGP+15] is generally regarded as the state-of-the-art solution, for both its efficiency and conceptual simplicity. In short, the Histogram Enumeration method approximates the probability density function of the key k with an histogram H having a limited number of bins. It is simply computed as the convolution of the histograms H_i built for each subkey, that is, $H = ((H_0 * H_1) * ...) * H_{m-1}$. Each bin contains the number of key candidates whose log probability $\log(p(k))$ falls in a given range. The real known key k^* will fall in one of the bins. The lower bound on the rank can be simply computed by summing the number of elements of each bin having a probability range higher than the real key. The higher bound can be simply computed by further adding the number of elements of the bin containing the real key. Clearly, the higher the number of bins, the tighter the bounds, the higher the complexity and execution time.

The Polynomial Rank Outlining Algorithm (PRO) [BLv15] solves the problem of estimating arbitrarily tight bounds for the key rank by observing its similarity with the problem of estimating arbitrarily tight bounds on the number of y-smooth integers (i.e., integers that are decomposed in primes less or equal that y) lower than a certain value, for which efficient solutions exist. Results are similar to [GGP+15].

After mapping floating point scores to integers with desired precision, the Key Rank method [MOOS15] formulates the key ranking problem as a counting knapsack problem, and solves it with path-counting in a directed acyclic graph. It is used in [MMOS16] to study the distribution of the rank over multiple experiments. It is mathematically equivalent to the histogram method [GGP+15], as proven by Martin et al. [MMO18].

A recent solution, named ESRank [DW19c, DW21], introduces the idea of approximating the probability distributions of subkeys using exponential sampling. That is, a given probability distribution $P[i]$ is approximated with $P[SI[i]]$, where SI is a set of indices such as $SI[i] = \lfloor \gamma SI[i-1] \rfloor$ and γ is a constant. For similarly tight bounds, ESrank achieves a good performance that is on-par with that of the Histogram Enumeration method, regarded in the paper as the best rank estimation algorithm to date [DW21]. An advantage of ESRank is that the tightness of its bounds can be chosen arbitrarily, while the tightness of the Histogram Enumeration method depends not only on the number of bins but also on the probability distribution.

PRank [DW19b] estimates only an upper bound on the rank via a closed-formula, by first approximating the probabilities of each subkey with a Pareto-like analytical function.

Grosso [Gro18] observed that key rank estimation algorithms do not scale well (or even do not scale at all) for large key size (e.g., RSA keys with up to 1024 bytes), and proposed an improvement of the Histogram Method that increases scalability. Both time and space complexity of the histogram convolution are reduced for large keys, while keeping a good tightness, by taking the following strategy. First, the final histogram for the entire key is constructed from the histograms of each subkey by recursively performing the convolution of pairs of histograms in a tree-like fashion. That is, the histograms H_i are regarded as the leaves of a binary tree and the final result is computed at the root by recursively performing a convolution at each node. Second, each time two histograms are convolved, a batching step is performed to merge bins two by two. This way, the number of bins after convolution remains constant (though their size doubles). At large key size this method performs better than the original histogram approach and was shown to work for keys up to 1024 bytes

(though the difference between upper and lower bound increases up to around 200 bits).

Choudary et al. [CP17, TCRP21] propose an efficient method to bound Massey's guessing entropy (GM) that scales well with large keys, though GM and rank are different metrics [Gro18]. Radulescu et al. [RPC22] show that GM is generally a lower bound to the empirical guessing entropy computed as the average rank over multiple experiments (each using [GGP+15]) or the Guessing Entropy Estimation Algorithm by Zhang et al. [ZDF20], which Young et al. [YMO22] outperform with classic rank estimation (based on [MOOS15]).

The methods that we have discussed generally assume that the probability distributions (given the side channel leakage) are independent for each subkey, as it is the case for AES-128 and RSA, but not for AES-256. In the specific case of template attacks on Diffie-Hellman key echange in Elliptic Curve Cryptography, Lange et al. [LvVW14] propose a key enumeration algorithm that can deal with non-independency, using a variation of Pollard's kangaroo methods. Wang et al. [WLW17] extend rank estimation to the case of Correlation-Enhanced Power Analysis Collision Attack (CECA) [MME10] on AES-128, where the attacker learns both the scores of individual subkeys $d(k_i)$ and the scores of subkey differences $d(k_{i,j}=k_i \oplus k_j)$. For each value $m \in [0,255]$ of k_0 and for each other subkey $j \in [1,15]$, the score $d(k_j)$ is replaced with $d(k_j)+d(k_{i,j}=m \oplus k_j)$, and the rank is computed. The final rank is the sum of all ranks. The process could be expanded to consider all values $n \in [0,255]$ of k_1, but enumerating further subkeys would quickly become computationally infeasible.

Early attempts at applying a statistical sampling (Monte Carlo) strategy All the rank estimation approaches that we have described compute deterministic bounds with a deterministic algorithm. One early attempt at using a statistical sampling (Monte Carlo) approach was proposed by Van Vredendaal [vV14]. It is based on the following idea. A number N of samples $k=(k_0,...,k_{m-1})$ is uniformly drawn from the space K of all possible keys, forming a sample S. For each sample k, the corresponding probability is computed as $p(k)=\prod_{i=0}^{m-1} p_i(k_i)$. For a sufficiently large sample size N, the ratio $|\{k' \in S | p(k') < k^*\}|/N$ converges to $|\{k' \in K | p(k') < k^*)\}|/|K| = R_{k^*}/|K|$. Hence, the rank can be estimated from the ratio between the number of key in the sample that have higher probability than the real key and the size of the sample. Unfortunately, it has been noted that such approaches may be applied to toy examples with small key spaces, but will require impractically large sample sizes to converge because, using real-world cryptographic parameters, $R_{k^*}/|K|$ can become vanishingly small, making it often unfeasible to have even a single sample in $\{k' \in S | p(k') < k^*\}$ [VGS13]. Nevertheless, Bernstein et al. [BLv15] still find a use of this method as a refinement technique to tighten the bounds computed with the seminal algorithm of Veryrat et al. [VGS13].

3 Monte Carlo Rank Estimation

In the following we present `MCRank`, a key rank estimation algorithm based on non-uniform statistical sampling. Initially based on the method proposed by Dell'Amico and Filippone [DF15] for the similar problem of password strength estimation [DMR10], `MCRank` addresses some challenges specific to key rank estimation. In particular, it uses automated rescaling of the input probabilities to deal with attacks that return scores or uncalibrated probabilities. In addition, it can be extended to deal with the interesting case of non-independent probability distributions which occurs with AES-256. `MCRank` is the first rank estimation method to effectively employ a Monte Carlo approach, as previous work generally computes deterministic bounds, while early attempts at using uniform sampling [vV14] would require impractical sample size to converge.

Algorithm 1 Rank estimation with independent probability distributions

procedure INDEPENDENT_MCRANK(p,n,k^*)
- ▷ $p_j(x)$: *(sampling) probability that a sample of subkey j has value x* ◁
- ▷ *In the simplest case, $p_j(x)=d_j(x)$ where $d_j(x)$ is the probability or normalized score returned by a side channel attack. See Algorithm 2 for a possible rescaling.* ◁
- ▷ n: *sample size* ◁
- ▷ k^*: *known key consisting of m t-bit subkeys $k^*_0,\ldots,k^*_{m-1}$* ◁
- ▷ i,j: *indices of the i-th sample and j-th subkey, respectively* ◁
- $P^* \leftarrow \prod_{j=0}^{m-1} p_j(k^*_j)$ ▷ *Probability of k^**
- ▷ *Matrix holding the cumulative sum of probabilities for each subkey* ◁
- **for** $j \in 0,\ldots,m-1$ **do**
  - **for** $x \in 0,\ldots,2^t-1$ **do**
    - $C_{j,x} \leftarrow \sum_{q=0}^{x-1} p_j(q)$
- $R \leftarrow 0$ ▷ *current value for the sum of Equation 1*
- $weaker \leftarrow 0$ ▷ *samples with probability higher than the real key*
- **for** $i \in 0,\ldots,n-1$ **do** ▷ *sample element i*
  - $P_i \leftarrow 1$ ▷ *probability of element i of our sample*
  - **for** $j \in 0,\ldots m-1$ **do** ▷ *subkey j*
    - $r \leftarrow$ random value uniformly taken from $[0,1)$
    - $k_j \leftarrow$ value of x such that $C_{j,x} \le r < C_{j,x+1}$ ▷ *value of the subkey*
    - $P_i \leftarrow P_i * p_j(k_j)$
  - **if** $P_i > P^*$ **then**
    - $R \leftarrow R+1/P_i$
    - $weaker \leftarrow weaker+1$
- ▷ *Return estimated rank and ratio of keys with higher score than k^* in the sample* ◁
- **return** $R/n, weaker/n$

3.1 Independent Probability Distributions

Let us consider the case of keys having size $m \cdot t$ bits, that we represent as m subkeys of t bits each. For example, in our AES-128 case, we have a key having 128 bits, split in $m=16$ subkeys of $t=8$ bit each. We represent a key k as a tuple of m subkeys: $k=(k_0,k_1,\ldots,k_{m-1})$; hence, the set of values a subkey can have is $T=\{0,\ldots,2^{t-1}\}$ and the set of all possible keys K is $K=T^m$.

Let K be the set of all possible keys; we consider a situation where an attacker has learned a scoring function $d_i : T \to [0,1]$ for each subkey k_i, where $\sum_{x \in T} d_i(x) = 1$.[1] We are currently considering the case of independent attacks on each subkey, so the global scoring function $d : K \to [0,1]$ for each key $k=(k_0,k_1,\ldots,k_{m-1}) \in K$ is simply computed as the product of all its subkeys: $d(k) = \prod_{i=0}^{m} d_i(k_i)$ as discussed in Section 2.

For a multi-set X (i.e., a generalization of a set allowing for duplicate elements), we define $W(X,k)$ as the multi-set of keys having score larger than $d(k)$,[2] i.e.,

$$W(X,k) = \{\{k' \in X | d(k') > d(k)\}\}$$

using the double brackets $\{\{\}\}$ to denote multi-sets. We formalize the rank estimation problem as efficiently finding the rank R_{k^*} of a known key $k^* \in K$, which is the number of keys having probability larger than the known key's score $d(k^*)$, i.e.,

$$R_{k^*} = |W(K,k^*)|.$$

[1] When the attack learns a probability distribution rather than a set of scores, we consider the probability distribution for a subkey to be the scoring function.

[2] In some pieces of work (e.g., Grosso [Gro18]) the rank definition considers keys that have score larger than *or equal to* $p(k)$. The difference between the definitions essentially does not matter for practical purposes, except for pathological cases in which large numbers of keys have the exact same rank.

If we choose X to be the set K of all keys (so that K and $W(K,k^*)$ do not have duplicates), then we can write $R_{k^*} = |W(K,k^*)| = |\{\{k' | d(k') > d(k^*)\}\}| = |\{k' | d(k') > d(k^*)\}|$) and this formulation is equivalent to the classic definition of rank presented in Section 2. However, the definition of $W(X,k)$ as a multi-set now also allows choosing X as a sample with replacement (i.e., with possible duplicates) of the keys, which will be useful to explain Monte Carlo methods.

As we have seen in Section 2, previously considered Monte Carlo approaches with uniform sampling [vV14] are impractical because they would need a very large sample size to converge and estimate the rank. The crucial difference that allows us to have a fast and practical Monte Carlo method is *non-uniform sampling*, in particular by *using the scores returned by the attack to generate a sample S of keys.* By sampling with higher probability keys that have a high score, we ensure that S will contain a much larger number of keys belonging to $W(K,k^*)$, thereby vastly improving convergence speed. We will then need to adopt a simple normalization mechanism to correct for the non-uniform sampling, i.e., dividing the weight of each sampled element by its probability to be sampled, with an approach that is reminiscent of well-known statistical techniques such as stratified sampling [HT52] and importance sampling [KvD78].

Our Monte Carlo method requires being able to generate a sample with replacement S of size $n = |S|$ such that the sampling probability $p : K \to [0,1]$ preserves the ordering induced by d, i.e., $\forall x,y \in K : p(x) > p(y) \iff d(x) > d(y)$. The original approach by Dell'Amico and Filippone only considers the case where $p = d$, i.e., the sampling probability of a given password is its score; this is effective when a method outputs well-calibrated probabilities (i.e., the probability that a key is k^* given the information observed from the side channel), but falls short when the models outputs a score that is not representative of probabilities or the probabilities are not calibrated. Allowing for $p \neq d$ allows us to efficiently handle a more generic use case through the *rescaling* technique described in the following. Since

$$W(X,k) = \{\{k' \in X | d(k') > d(k)\}\} = \{\{k' \in X | p(k') > p(k)\}\}$$

due to the order-preserving property defined above, modifying the sampling probability in this way does not affect our final result.

The estimated rank $\tilde{R}_{k^*}$ of the known key k^* is computed as

$$\tilde{R}_{k^*} = \frac{1}{n} \sum_{s \in S} \frac{[p(s) > p(k^*)]}{p(s)} = \frac{1}{n} \sum_{s \in W(S,k^*)} \frac{1}{p(s)} \tag{1}$$

using the Iverson bracket notation where [...] evaluates to 1 if it contains a true condition and 0 otherwise. Dell'Amico and Filippone [DF15] prove that this estimation is unbiased, i.e., the expected value of $\tilde{R}_{k^*}$ is indeed R_{k^*}; the proof comes from verifying that each key in $W(K,k^*)$ has an expected contribution of 1 to $\mathbb{E}(\tilde{R}_{k^*})$.

In Algorithm 1 we provide a simple pseudocode implementation of our approach, in which we sample non-uniformly from the set of possible keys. In essence, for each byte j we are given a sampling probability $p_j(x)$ that k_j is sampled with value x. As explained above, in the simplest case $p_j(x) = d_j(x)$ where $d_j(x)$ is the normalized score (or, ideally, probability) that k_j has value x according to the output of the side channel attack. To draw a sample x with probability $p_j(x)$, we first draw a sample r from a uniform distribution, and then we find x by inverting the cumulative sum of probabilities $C_{j,x}$. Repeating this for m subkeys generates the $i-th$ sample $k_i = (k_{i,0},...,k_{i,m-1})$ with probability $P_i = \prod_{j=0}^{m-1} p_j(k_j)$. We remark that it is not necessary to memorize the sampled keys; the only information needed to compute result from Equation 1 are the probabilities P_i for each item in the sample.

Estimation Error Analysis The estimation error is known to be $O(1/\sqrt{n})$; however, the expression provided in the original piece of work proving it [DF15] is unfeasible to explicitly evaluate, since it would require enumerating all keys in $W(K,k^*)$—the very task that rank estimation ought to accelerate. Here, we show how to compute confidence intervals. We

note that Equation 1 can be seen as the average of n independent estimates with sample size 1. Such estimates have finite variance (in fact, they can never be larger than $1/p(k)$ since elements in $W(S,k)$ have probability larger than $p(k)$). Because this is the average of independent estimates with finite variance, the central limit theorem applies: for $n \to \infty$ the errors follow a Gaussian distribution. We compute the Standard Error of Measurement (SEM) by taking the corrected sample standard deviation divided by the square root of n, that is:

$$SEM = \sqrt{\frac{\sum_{s \in S} \left(\tilde{R}_{k^*} - \frac{[p(s) > p(k^*)]}{p(s)} \right)^2}{n-1}} \cdot \frac{1}{\sqrt{n}}. \tag{2}$$

We can use this value to compute confidence intervals as usual (e.g., in 99.7% of the cases, $R_k \in \tilde{R}_k \pm 3\,SEM$). We should note that, even if we use sample sizes n that should be large enough to guarantee that our error distribution is Gaussian (e.g., $n \geq 1{,}000$), the error distribution might be extremely skewed. To double check that our confidence intervals are reliable, we validate them via the bootstrap percentile method [ET86] as well.

Evaluation Metrics We are interested in two key dimensions: tightness of the bounds and speed. We use the term uncertainty (as in measurement theory) as a synonym of tightness, as it is more appropriate for the probabilistic bounds provided by `MCRank`. We define it as:

$$u = \log_2 \frac{\text{upper bound}}{\text{lower bound}} = \log_2 \frac{\tilde{R}_{k^*} + 3\,SEM}{\tilde{R}_{k^*} - 3\,SEM}\ [bits] \tag{3}$$

It is equivalent to the definition of tightness used in existing work [VGS13]. Note that, the lower the uncertainty, the better. We measure speed as the runtime in seconds.

Unbalanced Sampling Policy and Rescaling The scores/probabilities d generated by a side channel attack might not be a well-calibrated estimation of the probability that a key is the real one given the observed side channel. This issue could happen, for example, when using correlation scores instead of the posterior probabilities of template attacks. More in general, for a profiled attack in a real-world setting, wrong assumptions or estimations errors when profiling the leakage model (e.g., wrongly assuming a linear model, or using a dataset where one class is under-represented) might render the model incapable of producing well-calibrated posterior probabilities, because the classes do not produce good predictions. In addition, the model estimated on one device could be sub-optimal for an attack on a different device and/or when aging or other environmental factors alter the victim's hardware. Finally, the output of some attacks cannot be easily translated in posterior probabilities (e.g., linear regression attacks that return the predicted Hamming Weight as a floating point number).

In such cases, a $p = d$ sampling policy in which scores are used as sampling probability might be unbalanced. That is, it might produce a sample in which there are prevalently keys with probability higher than the real key, or vice-versa prevalently keys with lower probability. This leads to poor convergence, requiring an extremely large sample size to reduce the uncertainty below an acceptable threshold. We solve this problem by (automatically) *rescaling* the sampling probabilities so that our sampling policy is balanced.

For some (positive) value of *delta*, we re-define the sampling probability as

$$p(k) = d(k)^\delta / \sum_{k'} d(k')^\delta \tag{4}$$

The rank is unaltered, as rescaling preserves the ordering $\forall x,y \in K : p(x) > p(y) \iff d(x) > d(y)$, but we can now choose δ to re-balance the sampling policy, as shown in Algorithm 2.

We start with $\delta = 1$, and loop until we find a suitable value for it. At every iteration, we create a small sample S_r with a sampling probability $p(k) = d(k)^\delta / \sum'_k d(k')^\delta$ and compute

the ratio r of keys in the sample that are weaker than k^*: $r = |W(S_r,k^*)|/|S|$. If the keys are reasonably well distributed between weaker and stronger than k^* (e.g., $0.05 \leq r \leq 0.95$), we are satisfied with the current value of δ. Otherwise, we update δ to a lower value if r is close to 1 (increasing the probability to sample lower-ranked keys) or to a higher value if, vice versa, r is close to 0 (increasing the probability to sample higher-ranked keys). Once the desired δ is found, rank estimation can be performed on a larger sample. We have found that this simple method is effective in adapting our sampling strategy to make `MCRank` converge quickly.

Excepting the corner cases where k^* is exactly the highest or lowest-score key, an appropriate value for δ with an acceptable expected value for r must exist. Consider that $\delta \to \infty$ results in $p(k) \to 1$ iff k is the highest-score key, and hence $r \to 1$; on the other end of our spectrum, as $\delta \to 0^+$ we move towards the uniform sampling which is suitable when the attack is essentially not informative about the key. Moreover, consider the expected ratio between the number of samples of a key k against those of k^*, i.e., $\frac{p(k)}{p(k^*)} = \left(\frac{d(k)}{d(k^*)}\right)^{\delta}$. It is apparent that, for increasing values of delta, the ratio $\frac{p(k)}{p(k^*)}$ increases smoothly for each key with higher score than k^* (score ratio larger than one) and conversely decreases for each key with lower score than k^* (score ratio lower than one). Of course probabilities must sum up to 1, so the expected value of r, i.e., the ratio of samples from $W(K,k^*)$, must grow monotonously as a function of δ. As a consequence of this, our algorithm eventually converges by creating a sample, evaluating r and accordingly updating δ if r is too small or too large, in a feedback loop fashion that stops at a suitable value for δ.

Algorithm 2 performs a grid search to find a suitable value for δ. A bad choice of the increment parameter could make it converge slowly (if too small) or not at all (too large); moreover, the randomness due to sampling could also in principle cause problems. In our experiments, we used a value of 0.1 for which we observed no such problems in practice; we leave an improvement of the algorithm (e.g., using binary search) to give better theoretical guarantees to further work. Another issue might arise when, due to very unlucky scenarios or buggy models, k^* is among the keys with *lowest* score. In that case, we can allow for $\delta < 0$ which essentially inverts the problem, because the algorithm returns the number of keys with $p(k) > p(k^*)$, that is those with score *lower* than k^* ($d(k) < d(k^*)$). In this case, the estimation of k^*'s rank is found by subtracting $|K|$ from the result.

Implementation Details Our own implementation is written in Python and it is accelerated (and made concise) thanks to the vectorization provided by the NumPy library [HMvdW+20]. It differs from the pseudocode described in Algorithm 1 for a few details: (i) we update the probabilities by inverting the order of the for loops: first iterating over subkeys and then sample elements, because this yields better performance (binary search to generate subkeys of many sample elements at once is parallelized); (ii) we represent probabilities as their logarithms (*log-probabilities*) to avoid numerical issues for very small probabilities and to accelerate computation because products of probabilities become sums of their logarithms. When working with very large RSA keys, the rank and some intermediate values in the calculation of the SEM might be too big to be represented with the IEEE 754 double-precision floating point values used in Python. To solve this problem, we use the `mpmath` library [J+13] to represent the rank with arbitrary precision, with minimal impact on performance as all other variables (e.g., probabilities) are not impacted.

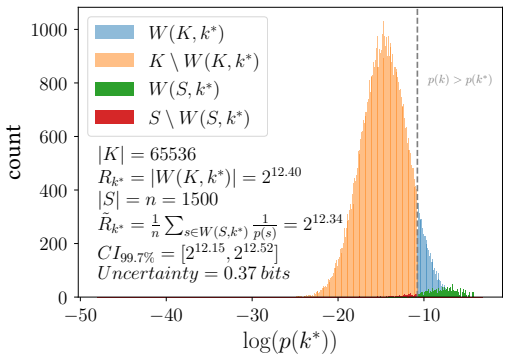
Visualization with Toy Example We show how `MCRank` works on a low-dimensional toy example. We consider an attack where the key is 2-byte long, mimicking an attack on a much smaller key space. We run a template attack on simulated traces to obtain the probabilities (Section 4 provides more details on how we simulate traces). Given the small size of the search space ($|K| = 65{,}536$), we can easily enumerate all the possible keys $k \in K$ and their probability $p(k)$. The rank of the known key k^* is simply given by the size of the set $W(K,k^*)$

Algorithm 2 Rank estimation with rescaling

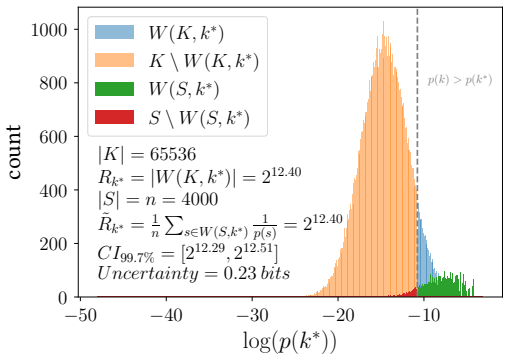
```
procedure RESCALING_MCRANK(d,n,k*,maxiter,threshold,increment,percentage)
    ▷ d_j(x): score indicating the likelihood that subkey j has value x ◁
    ▷ n: sample size ◁
    ▷ k*: known key consisting of m t-bit subkeys k*_0,...,k*_{m-1} ◁
    ▷ maxiter: maximum number of iterations for rescaling ◁
    ▷ threshold: maximum deviation from 50% balance ◁
    ▷ increment: increment/decrement step of the rescaling factor ◁
    ▷ percentage: percentage of samples on which rescaling is performed ◁
    δ ← 1                                                  ▷ Rescaling factor
    ▷ Run MCRank on a subset of samples until |r − 0.5| < threshold ◁
    while q < maxiter do
        ▷ Rescale the scores and normalize them so that they sum to 1 ◁
        for j ∈ 0,...m−1 do
            s ← 0
            for x ∈ 0,...2^t − 1 do
                p_j(x) ← d_j(x)^δ
                s ← s + p_j(x)
            for x ∈ 0,...2^t − 1 do
                p_j(x) ← p_j(x)/s
        ▷ r is the ratio of samples with higher score than k* ◁
        _,r ← Independent_MCRank(p,percentage·n,k*)
        if |r − 0.5| ≤ threshold then                      ▷ We found a suitable value for δ
            Break out of the while loop
        if r > 0.5 then
            ▷ Too many samples with higher score than k*: reduce δ ◁
            δ ← δ − increment
        else
            ▷ Not enough samples with higher score than k*: increase δ ◁
            δ ← δ + increment
        q ← q + 1
    ▷ Return estimated rank and ratio of keys with higher score than k* in the sample ◁
    return Independent_MCRank(p,n,k*)
```

of keys with probability larger than the known real key, that is: $R_{k^*} = |W(K,k^*)| = 2^{12.40}$. We then apply `MCRank` to this setting, extracting $n = |S| = 1{,}500$ samples and selecting those with higher probability than the real key (i.e., $W(S,k^*)$). Finally, we can estimate the rank $\tilde{R}_{k^*} = \frac{1}{n}\sum_{s \in W(S,k^*)} \frac{1}{p(s)} = 2^{12.34}$ with uncertainty 0.37 bits (the real rank falls in the 99.7 % confidence interval $[2^{12.15}, 2^{15.52}]$). Increasing the sample size to $n = |S| = 4{,}000$ improves the estimate to $\tilde{R}_{k^*} = 2^{12.40}$ with uncertainty 0.23 bits. In Figure 1 we compare the results of exhaustive enumeration and `MCRank`'s sampling for smaller (a) and larger (b) sample size, and with rescaling (c).

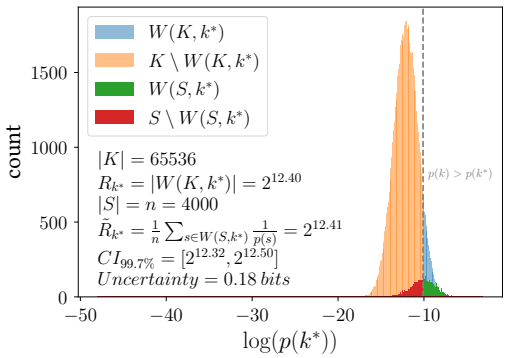
In the exhaustive search we can observe that $|W(K,k^*)| \gg |K \setminus W(K,k^*)|$. Instead, in all cases (a,b,c) the sample S used by `MCRank` is intentionally chosen to have samples falling close the real key, with $|W(S,k^*)|$ similar to $|S \setminus W(S,k^*)|$. This sampling policy, achieved by sampling based on the (rescaled) output probabilities/scores of the side channel attack, is what makes `MCRank` able to quickly converge and compute the rank using a small sample size $|S| \ll |K|$. When the sample size $|S|$ is bigger (b) the uncertainty is lower (i.e., better) than when the sample size is lower (a). In (a) and (b), $|W(S,k^*)|$ is not exactly equal to $|S \setminus W(S,k^*)|$, because the input probabilities are not perfectly balanced. To counter this problem, we rescale the probabilities by elevating them to a rescaling factor δ. The rescaling



(a) Smaller sample size



(b) Bigger sample size



(c) Bigger sample size and rescaling ($\delta \approx 0.52$)

Figure 1: Visualization of the `MCRank` method for a 2-byte key toy example. `MCRank` computes an accurate estimate of the rank of the known key k^* from a small subset S of the keys K, sampled according to their probability. The non-uniform sampling strategy results in fewer low-probability keys in the sample. Increasing the sample size from 1,500 (a) to 4,000 (b) reduces the uncertainty. The input probabilities can be rescaled to ensure $|W(S,k^*)| \approx |S \setminus W(S,k^*)|$ even if input probabilities were not balanced further reducing the uncertainty (c). Note that, no matter which rescaling we use for sampling, we are still computing the same rank.

factor is updated until $|W(S,k^*)| \approx |S \setminus W(S,k^*)|$ (happening at $\delta \approx 0.52$). As shown in (c), rescaling does not alter the rank because it preserves the ordering of the keys. However, the better balance results in a better (i.e., lower) uncertainty.

Algorithm 3 Rank estimation with dependent probability distributions

procedure DEPENDENT_MCRANK(p,n,k^*)
- ▷ $p_j(x)$: *probability that subkey j has value x (defined only for $j \in [0,m/2)$)* ◁
- ▷ n: *sample size* ◁
- ▷ k^*: *known key consisting of m t-bit subkeys $k^*_0,...,k^*_{m-1}$* ◁
- ▷ i,j: *indices of the i-th sample and j-th subkey, respectively.* ◁
- compute $p_{j,x|k^*_l}$ where $k^*_l = (k^*_0,...,k^*_{m/2-1})$
- $P^* \leftarrow \prod_{j=0}^{m/2-1} p_j(k^*_i) \prod_{j=m/2}^{m-1} p_{j,k^*_j|k^*_l}$
- **for** $j \in 0,...,m/2-1$ **do** ▷ *cumulative sum of probabilities for each subkey*
  - **for** $x \in 0,...,2^t-1$ **do**
    - $C_{j,x} \leftarrow \sum_{q=0}^{x-1} p_j(q)$
- $R \leftarrow 0$ ▷ *current value for the sum of Equation 1*
- **for** $i \in 0,...,n-1$ **do** ▷ *sample element i*
  - $P_i \leftarrow 1$ ▷ *probability of element i of our sample*
  - **for** $j \in 0,...m/2-1$ **do** ▷ *sample subkey j (left half)*
    - $r \leftarrow$ random value uniformly taken from $[0,1)$
    - $k_j \leftarrow$ value of x such that $C_{j,x} \leq r < C_{j,x+1}$ ▷ *value of the subkey*
    - $P_i \leftarrow P_i * p_j(k_j)$
  - compute $p_{j,x|k_l}$ where $k_l = (k_0,...,k_{m/2-1})$ ▷ *run a template attack based on k_l*
  - **for** $j \in m/2,...m-1$ **do** ▷ *sample subkey j (right dependent half)*
    - **for** $x \in 0,...,2^t-1$ **do**
      - $C_{j,x} \leftarrow \sum_{q=0}^{x-1} p_{j,q|k_l}$
    - $r \leftarrow$ random value uniformly taken from $[0,1)$
    - $k_j \leftarrow$ value of x such that $C_{j,x} \leq r < C_{j,x+1}$ ▷ *value of the subkey*
    - $P_i \leftarrow P_i * p_j(k_j)$
  - **if** $P_i > P^*$ **then**
    - $R \leftarrow R + 1/P_i$
- ▷ *Return estimated rank.* **return** R/n ◁

3.2 Dependent Probability Distributions

As we have discussed in Section 2, a major use case for `MCRank` is that of attacks that return non-independent probabilities for different bytes of the key. This happens for example in an algorithm like AES-256, because, as we have seen in Section 2, the attack on the second half of the key depends on the attack on the first half. We refer to this case as having dependent probability distributions among subkeys (where these probabilities are those returned by the attack given the leakage, and not the original probabilities with which bytes were drawn). This is a very challenging problem for existing rank estimation techniques such as the histogram approach, because directly adopting it would require repeating the template attack for each single possible value of the first (*left*) half of the key. For the AES-256 algorithm, the number of template attacks one would need to compute could be up to 2^{128}, which is clearly unfeasible. Conversely, we show that our approach only requires computing a number of template attacks proportional to the number of samples, which is much lower (i.e., $|S| << |K|$).

We can consider that the first template attack only gives us probability distributions for the left half of subkeys $p_0,...,p_{m/2-1}$. For a known left half of the key $k_l = (k_0,...,k_{m/2-1})$, we can however compute a new template attack giving us values $p_{j,x|k_l}$ for $j \in [m/2,m)$, representing the probability of subkey j having value x given that k_l is the left half of the key and given the measured value of the leakage L. In more formal terms,

$$p_{j,x|\hat{k}_l} = \mathbb{P}(k_j = x | k_l = \hat{k}_l, L) \quad \forall j \in \{m/2,...,m-1\}, x \in T.$$

The key difference with the independent-probabilities approach we have seen so far is that

here we will need to recompute the template attack based on each left half of a key we encounter–both for the known key k^* and for each key in the sample. In Algorithm 3, we present it in pseudocode. We can see that here we need $O(n)$ template attacks–one for the left half of k^*, and one for each element in the sample. In Section 4 we observe that the amount of computation due to recomputing the template probabilities dwarfs the rest of the process, increasing the runtimes we observe from a milliseconds to tens of seconds. In our implementation, we perform the additional optimization of generating all the first halves of the keys at once and group the ones that coincide, to avoid running multiple times a template attack for the same key.

4 Experimental Evaluation

`MCRank` is simple, fast (the execution time is $O(n)$ with the sample size n), accurate (the SEM is $O(1/\sqrt{n})$), and it scales well for very large keys (e.g., RSA 2048 bytes). It can also work with non-independent probability distributions (e.g., AES-256).

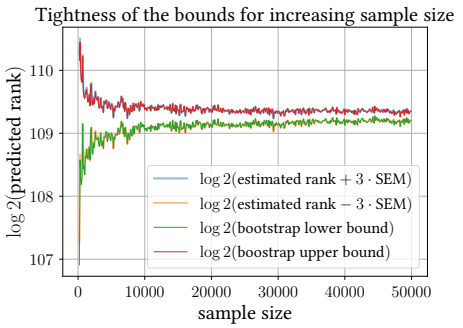
Baseline As explained in Section 2, the state-of-the-art references for comparison are the Histogram Enumeration method [GGP+15] for AES-128 and its extension for large keys [Gro18] for RSA. GMBounds [CP17] is also relevant for the RSA case, though it computes Massey's guessing entropy that is a different metric than the rank, so that the comparison is only qualitative. None of the previous algorithms deals with the case of dependent probabilities distributions occurring in the AES-256 case.

Setup We use our Python implementation (Section 3). For [GGP+15] we use a C++ implementation with a Python wrapper [Gro20]. The comparison is fair because: (i) the baseline uses the fastest language, (ii) execution time excludes the overhead of the Python wrapper, (iii) execution times are significantly different for AES-128, and (iv) for AES-256 [GGP+15] cannot even be applied because of non-independency. We write a Python wrapper (with similar considerations) for the C++ code of [Gro18] (obtained from the author) and for the Matlab implementation [Cho17] of [CP17]. We use simulated traces, with the leakage l following the Hamming Weight model plus Gaussian noise (i.e., $l(y) = HW[y] + n$, $n \sim N(\mu, \sigma^2)$). For AES-128 y is the output of the S_{box} at the first round. For AES-256, we simulate the first round to compute y as the output of the S_{box} at the second round. We implemented template attacks on AES-128 and AES-256. For each key byte, these attacks return a probability (in logarithm) for each of the 256 possible values. We also implemented a profiled correlation attack, which returns scores, to show how `MCRank` can handle this case, too. We also evaluate `MCRank` in the case of state-of-the-art Deep Learning attacks on real traces (ASCADv2 [MS21]). We run our experiments on an HP ENVY laptop (Intel(R) Core(TM) i7-4700MQ CPU @2.40GHz, 11GiB Memory, Ubuntu 22.04) in a Conda environment.

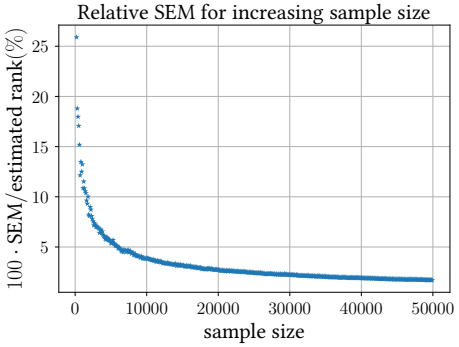
Uncertainty The uncertainty of `MCRank` can be tuned by controlling the sample size n, because the SEM is $O(1/\sqrt{n})$ as explained in Section 3.

> ***MCRank*** *rapidly achieves excellent accuracy (low uncertainty), as the confidence interval on the rank estimate quickly becomes tighter for increasing sample size.*

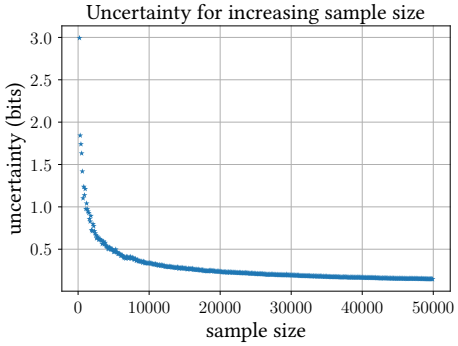
To show the uncertainty of `MCRank` we run a template attack on AES-128 with 5,000 template traces, 2 attack traces, and standard deviation of the noise in the simulated leakage equal to 2. To establish a ground truth we run the Histogram Enumeration method (with parameters $merge = 2$, $bins = 2{,}048$), obtaining that the rank lies in the interval $2^{109.17}$ to $2^{109.39}$. We then run `MCRank` for increasing sample size (200 to 50,000). Figure 2a shows how the 99.7 % confidence interval quickly becomes tighter as the sample size increases. The uncertainty follows a similar trend, shown in Figure 2c. With as few as 23,800 samples, the uncertainty of `MCRank`



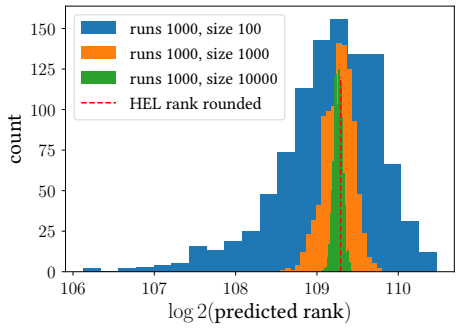
(a) Bounds (99.7 % confidence) vs. Sample Size



(b) Relative SEM (%) vs. Sample Size



(c) Uncertainty (99.7 % confidence) vs. Sample Size

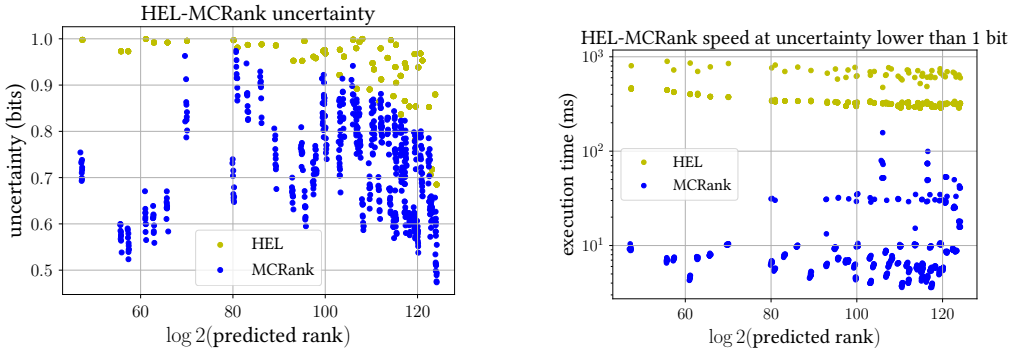


(d) Rank distribution (1000 runs) vs. Sample Size

Figure 2: AES-128. `MCRank` quickly converges to accurate rank estimates for increasing sample size. Confidence interval (a), relative SEM (b), and uncertainty (c) (i.e., logarithm of the ratio between upper and lower bound) rapidly become smaller. The confidence interval is computed from the SEM: under the assumption that the error is normally distributed (d), the rank falls in the $\pm 3\,SEM$ interval with 99.7 % confidence. Using the bootstrap method with $\alpha = 0.003$ provides the same bounds (a).

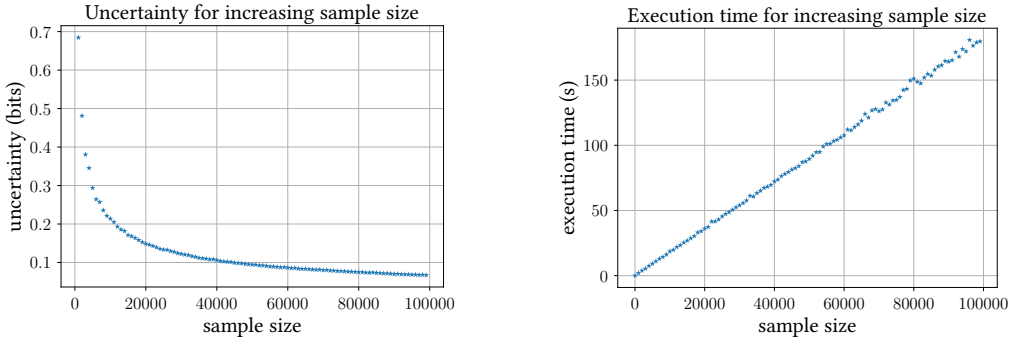
becomes lower (i.e., better) than that of the Histogram Enumeration method. The confidence interval is computed starting from the SEM, whose relative value is depicted in Figure 2b. As explained in Section 3, based on the central limit theorem we can assume that the error follows a normal distribution. Therefore, we compute the 99.7 % confidence interval as $\tilde{R}_k \pm 3\,SEM$. To further validate this assumption, we compute the upper and lower bounds of the confidence interval using the bootstrap percentile method [ET86] with $\alpha = 0.003$. As it can be seen in Figure 2a, the bounds computed with the two methods almost coincide. To visualize the uncertainty of `MCRank` from another perspective, in Figure 2d we draw the distribution of the rank estimate for multiple runs (1,000) for varying sample size (100, 1,000, 10,000). As the sample size increases the width of the distribution decreases (lower SEM, lower, i.e, better, uncertainty).

Execution Time The execution time of `MCRank` depends linearly on the sample size. This experimentally confirms our $O(n)$ computational complexity; higher ranks imply slightly more computation because the $W(S,k)$ set of Equation 1 becomes larger and thus requires operations on larger datasets. `MCRank`'s output is a probabilistic confidence interval, as opposed to the Histogram Enumeration method's certain bounds. We consider here a case that we deem equivalent to "almost certainty" in practical settings: to do so, we run 10 different template at-



(a) Uncertainty vs. log2(Rank) **(b)** Execution Time vs. log2(Rank)

Figure 3: AES-128. Comparison between Histogram Enumeration and `MCRank`. `MCRank` is orders of magnitude faster (b) while achieving comparable uncertainty (a).



(a) Uncertainty vs. Sample Size **(b)** Execution Time vs. Sample Size

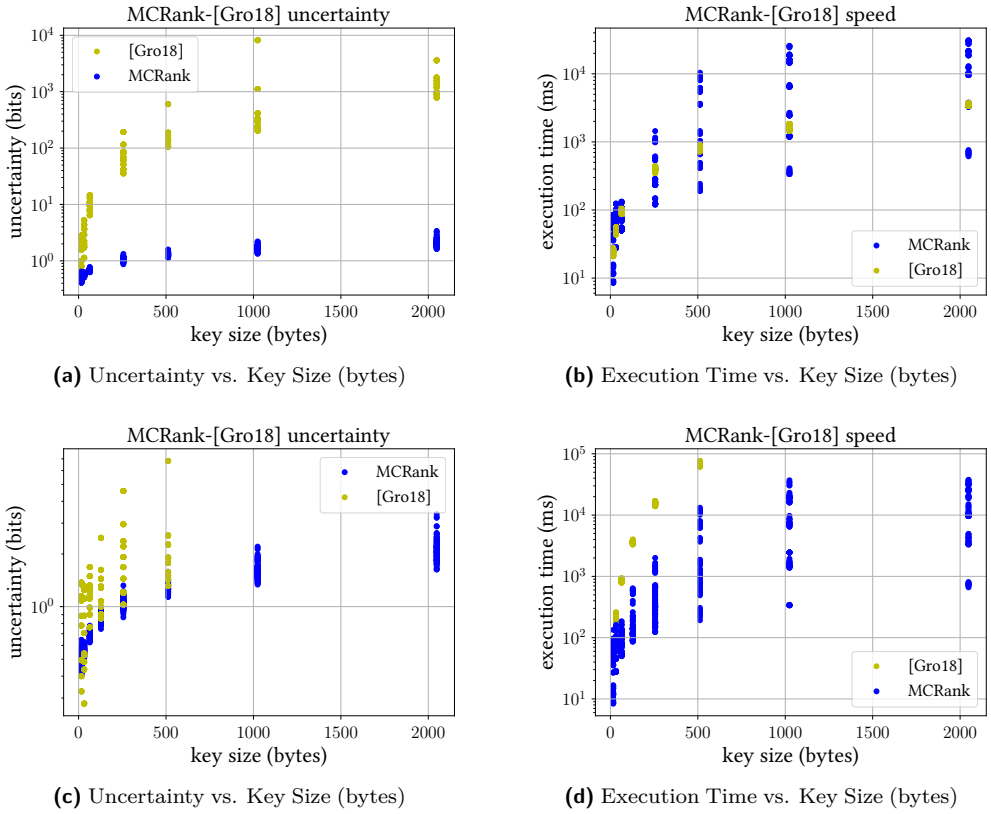
Figure 4: AES-256. Uncertainty (a) and execution time (b) for increasing sample size.

tacks, each for 6 different values of the noise variance, for a total of 60 cases. For each case, we run the Histogram Enumeration method (with $merge = 2$) increasing the number of bins until the uncertainty value becomes lower then 1 ($HEL\ Uncertainty < 1$). Then, we run `MCRank` for increasing $sample size$, until the uncertainty is lower (i.e., better) than that of the Histogram Enumeration method ($MCRank\ Uncertainty < HEL\ Uncertainty$) for 10 consecutive times.

> ***MCRank** is orders of magnitude faster than the Histogram Enumeration method, for comparable uncertainty.*

We demonstrate this with the experiment reported in Figure 3, after recording the uncertainty and execution time of both techniques for 10 times. As we can see, with uncertainty that is comparable for practical purposes, `MCRank` is orders of magnitude faster than the Histogram Enumeration method.

Probability distributions given the leakage non-independent among subkeys: One of the main advantages of `MCRank` is that it easily scales to scenarios in which: (i) the key is large; (ii) the probabilities of subkeys are not all independent; (iii) the search space is too large to be treatable with conventional techniques. We demonstrate this with the example of template attacks on AES-256 with known random plaintext and unknown key. In this case, the key is 32-byte long. Besides the size, the main difference with AES-128 is that the second half of the key is processed at the second round. As explained in Section 2 this requires first attacking the first



(a) Uncertainty vs. Key Size (bytes)

(b) Execution Time vs. Key Size (bytes)

(c) Uncertainty vs. Key Size (bytes)

(d) Execution Time vs. Key Size (bytes)

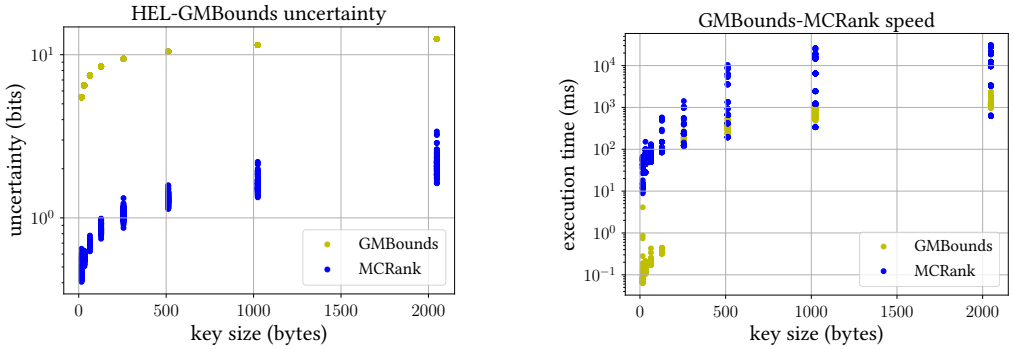
Figure 5: RSA key-load attack. Comparison with CARDIS18 [Gro18]. `MCRank` has lower (i.e., better) uncertainty (a) for similar execution speed (b). For similar accuracy (c), `MCRank` is faster (d). For keys larger than 512 bytes, CARDIS18 cannot achieve the same accuracy as `MCRank` because, due to the large number of bins, the program exceeds the memory limits and is killed by the operating system.

round and then using the result to attack also the second round. As a consequence, the probabilities of the last 16 bytes are dependent on the success of the first 16. Here, we pessimistically consider a perfect enumeration strategy that can still guess all keys by descending probability.

> *The statistical sampling approach of* `MCRank` *excels in the conditions of large guessing space and non-independent attack steps. Indeed,* `MCRank` *can compute a limited number of template attacks and still return an accurate estimate of the rank.*

`MCRank` requires at most $n+1$ template attacks where n is the sample size, which can be kept significantly smaller than the rank. Since sampling and ranking with `MCRank` are orders of magnitude times faster than a template attack, the execution time is dominated by the $O(n)$ template attacks needed, independently from the rank.

We repeat a similar evaluation as we did for AES-128. Results are similar, though the execution time moves from tens of milliseconds to up to a few minutes due to the cost of executing template attacks (at most, one per sample), as shown in Figure 4. We repeat the experiment for different ranks. A sample size of 10,000 is, as in the AES-128 case, enough to obtain an uncertainty of less than 0.5 bits for high ranks ranging from 2^{100} to 2^{240}, with an execution time of less than 25 s.



(a) Uncertainty vs. Key Size (bytes)

(b) Execution Time vs. Key Size (bytes)

Figure 6: Qualitative comparison between GMBounds [CP17] (Massey's guessing entropy) and `MCRank` (key rank) for the RSA key-load attack. `MCRank` has higher/similar execution time (b), but lower uncertainty (a) even with a small sample size (GMBounds' uncertainty cannot be controlled). Experiments are run for increasing key size and fixed sample size (1000). For large keys, [CP17] uses Matlab's symbolic values for variable precision, with a significant slowdown.

Very large keys: We now consider RSA key-load attacks, where each subkey is attacked independently, but where the number of subkeys can be very large. In general, key ranking algorithms are slow for large keys (see, for example, the comparison by Grosso [Gro18]). Recent work has improved scalability, optimizing the Histogram Enumeration method to obtain linear scaling with respect to the key size [Gro18], or taking the different approach of computing bounds for Massey's guessing entropy [CP17, TCRP21]. We compare `MCRank` with both approaches, but the comparison with [CP17, TCRP21] is only qualitative, because the Guessing Entropy is a different metric than the rank [Gro18]. Accuracy and execution speed for increasing number of key bytes are reported in Figure 5 and Figure 6. We are the first to show results for keys up to 2048 bytes (previous work stops at 1024 bytes). For similar execution time (Figure 5b), the uncertainty of `MCRank` is orders of magnitude smaller and grows slower than that of [Gro18] (Figure 5a). To achieve the same low (i.e., good) uncertainty as `MCRank` (Figure 5c), the number of bins in [Gro18] has to be increased with the key size, leading to a higher execution time, or even leading the process to be killed by the operating system because of an excessive memory consumption (Figure 5d). For large key size, `MCRank`'s accuracy is one order of magnitude lower (i.e., better) than [CP17] for similar execution time (Figure 6). For lower key size [CP17] does not have to use symbolic variables with arbitrary precision and can be faster.

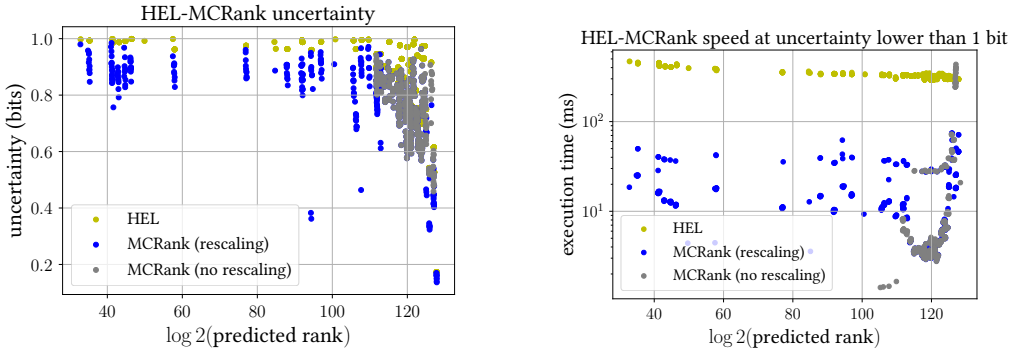
> *`MCRank` can easily scale to 2048-byte keys achieving much lower (i.e., better) uncertainty than previous work for similar execution time. Previous work is slower (or cannot execute due to excessive memory consumption) for similar uncertainty.*

In all these experiments, `MCRank` uses automatic rescaling on 100 % of the sample size, to keep the percentage of samples with probability higher than the real key in the 5 %–95 % range.

Scores instead of probabilities: To show `MCRank`'s ability to work with scores, we compare again with [GGP+15] using profiled correlation attacks. We use `MCRank` without rescaling and with automatic rescaling on 100 % of the sample size to keep the percentage of samples with higher probability than the real key in the 25 %–75 % range.

> *With rescaling, `MCRank` works also with scores instead of probabilities.*

Results are shown in Figure 7. Despite rescaling, `MCRank` is still faster than the Histogram Enumeration method for similar or lower (i.e., better) uncertainty. Without rescaling (gray),



(a) Uncertainty vs. log2(Rank)

(b) Execution Time vs. log2(Rank)

Figure 7: AES-128 with profiled correlation attack. Comparison between Histogram Enumeration and `MCRank`. `MCRank` is faster (b) while achieving comparable uncertainty (a), as long as rescaling is used (blue). Rescaling is necessary (blue vs. gray).

Table 1: Comparison with State-Of-The-Art (SOTA).

Case	Traces	Attack	Challenge	SOTA	`MCRank`	Results
AES-128	Sim.	Templates	Classic problem	[GGP+15]	Faster	Figure 3
AES-128	Sim.	Profiled Correlation	Unbalanced scores	[GGP+15]	Faster	Figure 7
AES-128	Real	Deep Learning	Real-world	[GGP+15]	Faster	Figure 8
AES-256	Sim.	Templates	Non independency	Unsolved	First to solve Fast	Figure 4
RSA 16-2048 bytes	Sim.	Templates	Scalability	[Gro18]	First to 2048 Faster	Figure 5

Faster: faster execution for same uncertainty, lower uncertainty in same execution time.

`MCRank` is not able to achieve the desired accuracy because of a high imbalance in the sampling, but rescaling (blue) solves the problem.

Deep-learning attacks on real traces: We evaluate `MCRank` with two Deep Learning attacks on the real traces of a protected implementation provided in ASCADv2 (see Section 2) and compare with Histogram Enumeration, showing higher speed for similar accuracy.

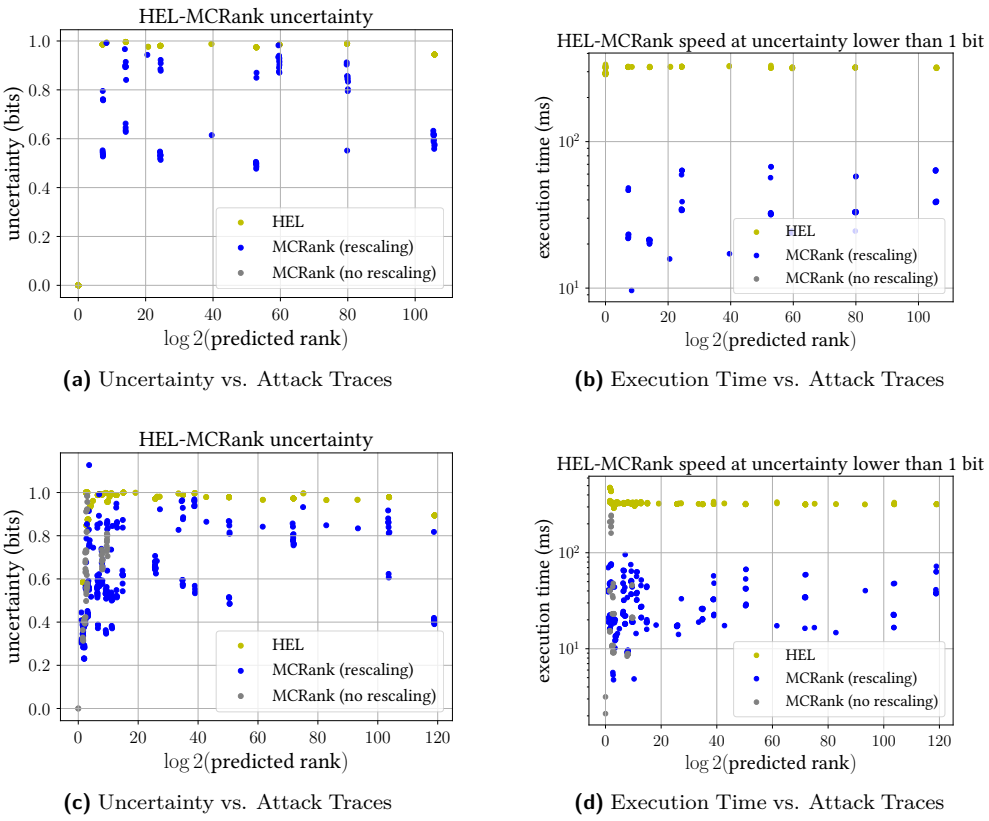
> *`MCRank` is faster than previous work also when evaluated with modern Deep Learning attacks on a real protected implementation, provided rescaling is used.*

Results are shown Figure 8 with both no rescaling and rescaling on 2 % of the samples to keep the number of samples with higher probability than the real key in the 5 %–95 % range.

Summary: Table 1 compares `MCRank` to previous work. `MCRank` is faster than [GGP+15] for AES-128, it is the first approach to solve the case of dependent probability distributions (found, e.g., in AES-256), and it scales better than [Gro18] for large keys, even reaching 2048 bytes.

5 Conclusion

We have presented `MCRank`, an approach to rank estimation based on Monte Carlo sampling. `MCRank` provides an unbiased estimate of the rank with a probabilistic confidence interval on its bounds. The tightness of the confidence interval quickly increases with the sample



(a) Uncertainty vs. Attack Traces

(b) Execution Time vs. Attack Traces

(c) Uncertainty vs. Attack Traces

(d) Execution Time vs. Attack Traces

Figure 8: Deep-learning attacks on real traces of a protected implementation of AES-128 (ASCADv2) for increasing number of attack traces. `MCRank` is faster (b,d) for similar or lower (i.e., better) uncertainty (a,c) than Histogram Enumeration also in this case. (a,b) and (c,d) correspond to two different attacks presented in [MS21] (with and without knowledge of the permutation index). With rescaling (blue) `MCRank` achieves the desired accuracy below 1 bit, while without rescaling (gray) this is often not possible (missing gray points).

size n (the SEM is $O(1/\sqrt{n})$), whereas execution time is only linear with it ($O(n)$). As a result, `MCRank` is orders of magnitude faster than the state-of-the-art Histogram Enumeration method in the case of AES-128. This was demonstrated with template and profiled correlation attacks on simulated traces, and deep-learning attacks on real traces from a protected implementation. The gains in simplicity and speed justify the use of probabilistic bounds. Most importantly, the sampling approach allows `MCRank` to treat for the first time the case of subkeys whose probability distributions given the side channel leakage are not independent, as it happens with AES-256. The same approach could be applied in the future to other similarly challenging cases in public key cryptography. `MCRank` scales well also for large keys (with independent attacks on each subkey), as it happens for key-load attacks on RSA. For the first time, we report excellent accuracy and low execution time for keys up to 2048 bytes. When necessary, `MCRank` rescales the input scores/probabilities to provide good accuracy even if they are unbalanced. While the rescaling algorithm that we propose is automatic, it requires selecting a few hyperparameters (e.g., the threshold on the balance of the samples). In conclusion, we have shown that Monte Carlo sampling is a profitable approach to key rank estimation. `MCRank` is available at https://github.com/giocamurati/mcrank.

Acknowledgement The authors are very grateful to the reviewers, and in particular to the Shepherd Dr. Aron Gohr, for their extensive contribution in improving the paper.

References

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer, Heidelberg, August 2003.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004.

[BLv15] Daniel J. Bernstein, Tanja Lange, and Christine van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. Cryptology ePrint Archive, Report 2015/221, 2015. https://eprint.iacr.org/2015/221.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020.

[Cho17] Marios O. Choudary. gmbounds, 2017. https://gitlab.cs.pub.ro/marios.choudary/gmbounds.

[CP17] Marios O. Choudary and P. G. Popescu. Back to Massey: Impressively fast, scalable and tight security evaluation tools. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 367–386. Springer, Heidelberg, September 2017.

[CPS16] Marios O. Choudary, Romain Poussier, and François-Xavier Standaert. Score-based vs. probability-based enumeration - A cautionary note. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *INDOCRYPT 2016*, volume 10095 of *LNCS*, pages 137–152. Springer, Heidelberg, December 2016.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

[DF15] Matteo Dell'Amico and Maurizio Filippone. Monte Carlo strength evaluation: Fast and reliable password checking. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 158–169. ACM Press, October 2015.

[DMR10] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, pages 983–991. IEEE, 2010.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 240–262. Springer, Heidelberg, May 2016.

[DW19a] Liron David and Avishai Wool. Context aware password guessability via multi-dimensional rank estimation. *CoRR*, abs/1912.02551, 2019.

[DW19b] Liron David and Avishai Wool. Fast analytical rank estimation. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 168–190. Springer, Heidelberg, April 2019.

[DW19c] Liron David and Avishai Wool. Poly-logarithmic side channel rank estimation via exponential sampling. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 330–349. Springer, Heidelberg, March 2019.

[DW21] Liron David and Avishai Wool. Rank estimation with bounded error via exponential sampling. *IACR Cryptol. ePrint Arch.*, page 313, 2021.

[ET86] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1(1):54–75, 1986.

[GGP+15] Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 117–129. Springer, Heidelberg, March 2015.

[Gro18] Vincent Grosso. Scalable key rank estimation (and key enumeration) algorithm for large keys. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, volume 11389 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2018.

[Gro20] EURECOM S3 Group. Python HEL, 2020. https://github.com/giocamurati/python_hel.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 444–461. Springer, Heidelberg, August 2014.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nat.*, 585:357–362, 2020.

[HT52] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

[J+13] Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*, December 2013. http://mpmath.org/.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KvD78] T. Kloek and H. K. van Dijk. Bayesian estimates of equation system parameters: An application of integration by monte carlo. *Econometrica*, 46(1):1–19, 1978.

[LvVW14] Tanja Lange, Christine van Vredendaal, and Marnix Wakker. Kangaroos in side-channel attacks. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 104–121. Springer, 2014.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, Heidelberg, August 2010.

[MMO18] Daniel P. Martin, Luke Mather, and Elisabeth Oswald. Two sides of the same coin: Counting and enumerating keys post side-channel attacks revisited. In Nigel P. Smart, editor, *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, volume 10808 of *Lecture Notes in Computer Science*, pages 394–412. Springer, 2018.

[MMOS16] Daniel P. Martin, Luke Mather, Elisabeth Oswald, and Martijn Stam. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 548–572. Springer, Heidelberg, December 2016.

[MOOS15] Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 313–337. Springer, Heidelberg, November / December 2015.

[MS21] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected AES implementation on ARM. *IACR Cryptol. ePrint Arch.*, page 592, 2021.

[Pub01] NF Pub. Specification for the advanced encryption standard (AES). *Tech. Rep. FIPS PUB 197, Federal Information Processing Standards*, 2001.

[RPC22] Anca Radulescu, Pantelimon George Popescu, and Marios O. Choudary. GE vs GM: efficient side-channel security evaluations on full cryptographic keys. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):886–905, 2022.

[TCRP21] Andrei Tanasescu, Marios O. Choudary, Olivier Rioul, and Pantelimon George Popescu. Tight and scalable side-channel attack evaluations through asymptotically optimal massey-like inequalities on guessing entropy. *Entropy*, 23(11):1538, 2021.

[VGRS13] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 390–406. Springer, Heidelberg, August 2013.

[VGS13] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 126–141. Springer, Heidelberg, May 2013.

[vV14] Christine van Vredendaal. Rank estimation methods in side channel attacks. Master's thesis, Eindhoven University of Technology, 2014.

[WLW17] Shuang Wang, Yang Li, and Jian Wang. A new key rank estimation method to investigate dependent key lists of side channel attacks. In *2017 Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2017, Beijing, China, October 19-20, 2017*, pages 19–24. IEEE Computer Society, 2017.

[Wur19] Antoine Wurcker. Ease of side-channel attacks on AES-192/256 by targeting extreme keys. Cryptology ePrint Archive, Report 2019/340, 2019. https://eprint.iacr.org/2019/340.

[YMO22] Rebecca Young, Luke Mather, and Elisabeth Oswald. Comparing key rank estimation methods. Cryptology ePrint Archive, Paper 2022/1282, 2022. https://eprint.iacr.org/2022/1282.

[ZDF20] Ziyue Zhang, A. Adam Ding, and Yunsi Fei. A fast and accurate guessing entropy estimation algorithm for full-key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):26–48, 2020.