# PRank: Fast Analytical Rank Estimation via Pareto Distributions

Liron David[1], Avishai Wool[2]

School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel
[1]lirondavid@gmail.com, [2]yash@eng.tau.ac.il

**Abstract.** Rank estimation is an important tool for a side-channel evaluations laboratories. It allows estimating the remaining security after an attack has been performed, quantified as the time complexity and the memory consumption required to brute force the key given the leakages as probability distributions over $d$ subkeys (usually key bytes). These estimations are particularly useful where the key is not reachable with exhaustive search. We propose a new method called PRank for rank estimation, that is conceptually simple, and more time and memory efficient than previous proposals. Our main idea is to bound each subkey distribution by a Pareto-like function: since these are analytical functions, we can then estimate the rank by a closed formula. We evaluated the performance of PRank through extensive simulations based on two real SCA data corpora, and compared it to the currently-best histogram-based algorithm. We show that PRank gives a good rank estimation with much improved time and memory efficiency, especially for large ranks: For ranks between $2^{80} - 2^{100}$ PRank estimation is at most 10 bits above the histogram rank and for ranks beyond $2^{100}$ the PRank estimation is only 4 bits above the histogram rank—yet it runs faster, and uses negligible memory. PRank gives a new and interesting method to solve the rank estimation problem based on reduction to analytical functions and calculating one closed formula hence using negligible time and space.

## 1 Introduction

### 1.1 Background

Side-channel attacks (SCA) represent a serious threat to the security of cryptographic hardware products. As such, they reveal the secret key of a cryptosystem based on leakage information gained from physical implementation of the cryptosystem on different devices. Information provided by sources such as timing [15], power consumption [14], electromagnetic emulation [20], electromagnetic radiation [2, 12] and other sources, can be exploited by SCA to break cryptosystems.

A security evaluation of a cryptographic device should determine whether an implementation is secure against such an attack. To do so, the evaluator needs to determine how much time, what kind of computing power and how

much storage a malicious attacker would need to recover the key given the side-channel leakages. The leakage of cryptographic implementations is highly device-specific, therefore the usual strategy for an evaluation laboratory is to launch a set of popular attacks, and to determine whether the adversary can break the implementation (i.e., recover the key).

Most of the attacks that have been published in the literature are based on a "divide-and-conquer" strategy. In the first "divide" part, the cryptanalyst recovers multi-dimensional information about different parts of the key, usually called subkeys (e.g., each of the $d = 16$ AES key bytes can be a subkey). In the "conquer" part the cryptanalyst combines the information all together in an efficient way via key enumeration [18, 22, 9]. In the attacks we consider in this paper, the information that the SCA provides for each subkey is a probability distribution over the $N$ candidate values for that subkey, and the SCA probability of a full key is the product of the SCA probabilities of its $d$ subkeys.

A security evaluator knows the secret key and aims to estimate the number of decryption attempts the attacker needs to do before he reaches to the correct key, assuming the attacker uses the SCA's probability distribution. Clearly enumerating the keys in the optimal SCA-predicted order is the best strategy the evaluator can follow. However, this is limited to the computational power of the evaluator. This is a worrying situation because it is hard to decide whether an implementation is "practically secure". For example, one could enumerate the $2^{50}$ first keys for an AES implementation without finding the correct key, and then to decide that the implementation is practically secured because the attacker needs to enumerate beyond $2^{50}$ number of keys. But, this does not provide any hint whether the concrete security level is $2^{51}$ or $2^{120}$. This makes a significant difference in practice, especially in view of the possibility of improved measurement setups, signal processing, information extraction, etc., that should be taken into account for any physical security evaluation, e.g., via larger security margins.

In this paper, we introduce a new method to estimate the rank of a given secret key in the optimal SCA-predicted order. Our algorithm enjoys simplicity and much improved time and memory efficiency.

***The rank estimation problem:*** Given $d$ independent subkey spaces each of size $N$ with their corresponding probability distributions $P_1, ..., P_d$ such that $P_i$ is sorted in decreasing order of probabilities, and given a key $k^*$ indexed by $(k_1, ..., k_d)$, let $p^* = P_1(k_1) \cdot P_2(k_2) \cdot ... \cdot P_d(k_d)$ be the probability of $k^*$ to be the correct key. The evaluator would like to estimate the number of full keys with probability higher than $p^*$, when the probability of a full key is defined as the product of its subkey's probabilities.

In other words, the evaluator would like to estimate $k^*$'s rank: the position of the key $k^*$ in the sorted list of $N^d$ possible keys when the list is sorted in decreasing probability order, from the most likely key to the least. If the dimensions, or $k^*$'s rank are small, one can easily compute the rank of the correct key by a strait forward key enumeration. However, for a key with a high rank $r$ the optimal-order key enumeration requires $\Omega(r)$ time which may be prohibitive, and the best currently-known optimal-order key enumeration algorithms require

$\Omega(N^{d/2})$ space, which again may be prohibitive. Hence developing fast and low-memory algorithms to estimate the rank without enumeration is of great interest.

### 1.2 Related work

The best key enumeration algorithm so far, in terms of optimal-order, was presented by Veyrat-Charvillon, Gérard, Renauld and Standaert in [22]. However, its worst case space complexity is $\Omega(N^{d/2})$ when $d$ is the number of subkey dimensions and $N$ is the number of candidates per subkey - and its space complexity is $\Omega(r)$ when enumerating up to a key at rank $r \leq N^{d/2}$. Thus its space complexity becomes a bottleneck on real computers with bounded RAM in realistic SCA attacks.

Since then several near-optimal key enumeration were proposed [6, 16, 19, 24, 4, 13, 9]. However, none of these key enumeration algorithms enumerate the whole key space within a realistic amount of time and with a realistic amount of computational power, hence the need for efficient and accurate rank estimation for keys that have a high rank.

The first rank estimation algorithm was proposed by Veyrat-Charvillon et al. at Eurocrypt 2013 [23]. They suggested to organize the keys by sorting their subkeys according to the a-posteriori probabilities provided, and to represent them as a high-dimensional dataspace. The full key space can then be partitioned into two volumes: one defined by the key candidates with probability higher than the correct key, one defined by the key candidates with probability lower than the correct key. Using this geometrical representation, the rank estimation problem can be stated as the one of finding bounds on these "higher" and "lower" volumes. It essentially works by carving volumes representing key candidates on each side of their boundary, in order to progressively refine the (lower and upper) bounds on the key rank. Refining the bounds becomes exponentially difficult at some point.

A number of works have investigated solutions to improve upon [23]. In particular, Glowacz et al. presented a more efficient rank estimation tool, that is based on a convolution of histograms and allows obtaining tight bounds for the key rank of (even large) keys [13]. The space complexity of their algorithm is $O(dB \log B)$ where $d$ is the number of dimensions and $B$ is a design parameter controlling the number of the histogram bins. The accuracy of their bounds depends on $B$. A comparable result was developed independently by Bernstein et al. [4]. In parallel, Ye et al. investigated an alternative solution based on a weak Maximum Likelihood (wML) approach [24], rather than a Maximum Likelihood (ML) one for the previous examples. They additionally combined this wML approach with the possibility to approximate the security of an implementation based on "easier to sample" metrics, e.g., starting from the subkey Success Rates (SR) rather than their likelihoods. Later Duc et al. described a simple alternative to the algorithm of Ye et al. and provided an "even easier to sample" bound on the subkey SR, by exploiting their formal connection with a Mutual Information metric [10].

Martin et al. [16] used a score-based rank enumeration, rather than a probability based rank estimation. They mapped the rank estimation to a knapsack problem, which can be simplified and expressed as path counting. They used additive scoring (the scores of different subkeys are added to score a full key), without specifying the relation between the scores and the probabilities. This makes it difficult to compare apples with apples: the quality of their rank estimation would have been comparable to other rank estimation only if they had used log-probabilities (whose addition is semantically equivalent to multiplication of probabilities).

Choudary et al. [8] presented a method for estimating Massey's guessing entropy (GM) which is the statistical expectation of the position of the correct key in the sorted distribution. Their method allows to estimate the GM within a few bits. However, our data shows that the *actual* guessing entropy, i.e., the *rank* of the correct key, is sometimes quite different from the expectation. Our work uses different methods for getting closed formula bounds to estimate the actual rank within a few bits.

### 1.3 Contribution

We start by investigating the prediction quality of SCA-based probability distribution functions on the data corpus of [17]. Our first contribution is that we show the SCA probability is overly optimistic; the predicted Guessing Entropy is much lower than the average true rank. Moreover, the true rank distribution has a long tail: large ranks do appear with non-negligible probability. We show that a Pareto distribution is a good model for the true rank distribution.

Motivated by this observation, we use Pareto-like functions to upper-bound the empirical SCA-based probability distributions. We first prove that one can always upper-bound a sorted probability distribution $P$ by a Pareto-like function $f$ that is anchored at 2 indexes at which $f(x) = P[x]$. We then fully characterize such upper-bounding functions, prove that there can only be $O(N)$ of them, and develop an efficient algorithm to find them.

Since Pareto-like functions are analytical, we use them to develop a new upper bound on the rank of a given key, as an explicit *closed formula*. Combined with the algorithm to find the upper-bounding Pareto-like functions, we obtain an $O(dN^2)$ rank upper-bound estimation algorithm we call PRank. We also developed a simple and efficient local-search rank lower-bound algorithm.

We evaluated the performance of PRank through extensive simulations based on two SCA data corpora of [17] and of [11], and compared it to the currently-best histogram rank estimation algorithm of [13]. We show that PRank gives a good rank estimation with much improved time and memory efficiency, especially for large ranks: For ranks between $2^{80} - 2^{100}$ PRank estimation is at most 10 bits above the histogram rank and for ranks beyond $2^{100}$ the PRank estimation is only 4 bits above the histogram rank—yet it runs faster, and uses negligible memory. PRank gives a new and interesting method to solve the rank estimation problem based on reduction to analytical functions and calculating one closed

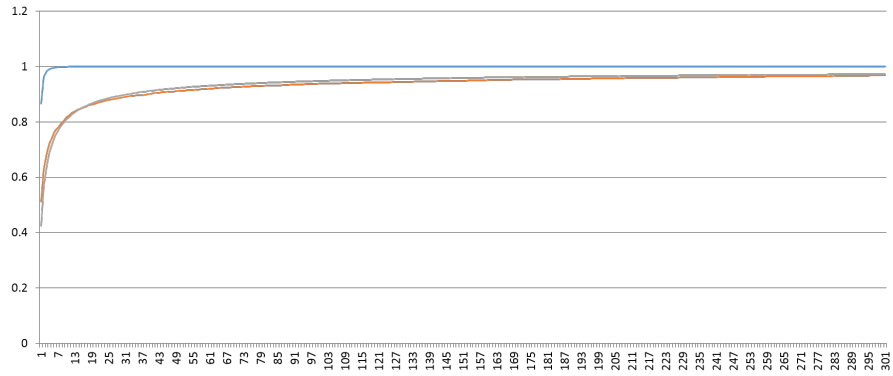


**Fig. 1.** The cdf of average SCA-predicted probability distribution $P_i$ (blue), true-probability distribution $P_i^*$ (red) and Pareto distribution (green) versus the true rank.

formula hence using negligible time and space. It is therefore a useful addition to the SCA evaluator's toolbox.

## 2 Motivation for the PRank algorithm

### 2.1 The Data Corpus

Probabilistic side channel attacks such as template attacks [7] produce a probability distribution for each subkey, interpreted as the probability that a particular subkey value is correct. In our study, we used the data of Oren, Weisse and Wool in [17]. Within this data, there are 936 probability distribution sets gathered from a specific SCA. The SCA of [17] was against AES [1] with 128-bits keys. The attack grouped the key bits into 4 32-bit subkeys, and hence its output probability distributions are over these 32-bit values. Each set in the corpus, consists of the correct secret key and 4 distributions, one per sub-key. The distributions are sorted in non-increasing order of probability. The SCA of [17] discards subkey candidates it finds to be unacceptable hence the probability distributions all have much fewer than $2^{32}$ values: the distribution length $N$ is at most $2^{17}$. This means that the largest rank that the data could predict is $N^4 \approx 2^{68}$.

### 2.2 Modeling the SCA-Distributions by Pareto Distributions

Assume that the SCA-distribution $P_i$ for a subkey $x_i$, is sorted in decreasing order, hence $P_i[j]$ is the SCA probability that the correct value for subkey $x_i$ has rank $j$. However, for the data corpus we also know the true rank $r^*$ of the correct value for each subkey $x_i$, according to the order implied by $P_i$. Thus we can calculate the true rank probability distribution: let $P_i^*[r]$ be the frequency

of subkey values such that the subkey at position $r$ according to the SCA-order is the correct subkey.

Figure 1 shows the average SCA-predicted probability distribution $P_i$ (averaged over all $936 \times 4$ distributions) versus the true rank distribution $P_i^*$, using the data of [17].

The figure shows that the SCA distribution is too optimistic: the cumulative distribution function (cdf) of $P_i$ grows much faster than that of $P_i^*$. This also leads to a significant difference in the guessing entropy [21], (also called Massey's guessing entropy [8]) with $G(X) = 1.2$ using the SCA-predicted distribution compared to $G(X) = 135$ for the true rank probability distribution $P_i^*$. This also means that using the terminology of [8], for the SCA in [17] there is a significant gap between the GM and the GE.

Further, we can see that the true distribution has a long tail: large ranks do appear with non-negligible probability. A good distribution which models long-tail distributions is the Pareto distribution [3]. If $X$ is a random variable with a Pareto distribution parameterized by constants $a > 0$ and $\alpha > 0$, then the pdf is given by $f_X(x) = a/x^\alpha$, for $x \geq 1$.

The usual definition of a Pareto distribution requires $\alpha > 1$, or even $\alpha > 2$ (so the expectation $E(X)$ is well defined), and $a$ is set to ensure $\int f_X(x)dx = 1$.

In order to model the true rank distribution by a Pareto distribution, we wish to find parameters that lead its expectation to equal the guessing entropy of the true rank distribution, 135, by solving the following equations:

$$\sum_{x=1}^{N} f_X(x) = \sum_{x=1}^{N} \frac{a}{x^\alpha} = 1$$

and

$$E(X) = \sum_{x=1}^{N} x \cdot f_X(x) = \sum_{x=1}^{N} \frac{a}{x^{\alpha-1}} = 135.$$

By local search we solve for $\alpha$ and the normalization coefficient $a$, and we obtain $\alpha = 1.575$, and $a = 0.424$. Figure 1 also shows a cdf curve for this Pareto distribution demonstrating the fit: it is almost completely obscured by the true distribution curve.

This observation has led us to our new method to estimate the rank of a given key. Pareto distributions have convenient analytical properties that allow closed formulas, and as we have seen, seem to offer good models for the real distributions.

## 3 Upper Bounds on the Rank

Throughout the paper we always assume that the probability distributions $P_i$ are sorted in decreasing order: e.g., $P_i[1]$ is the probability of the most likely value for subkey $i$. For notational convenience when we discuss a key $k = (k_1, .., k_d)$

we mean that $k_i$ is the rank, in the sorted distribution $P_i$, of the relevant subkey value.

**Definition 1** (*Rank*($k^*$))**.** *Let $d$ non-increasing subkey probability distributions $P_i$ for $1 \leq i \leq d$ and the correct key $k^* = (k_1, ..., k_d)$ be given. Let $p^* = P_1[k_1] \cdot ... \cdot P_d[k_d]$ be the probability of the correct key. Then, define Rank($k^*$) to be the number of keys $(x_1, ..., x_d)$ s.t. $P_1[x_1] \cdot ... \cdot P_d[x_d] \geq p^*$.*

### 3.1 The Box Bound

Given the correct key $k^*$, we start with an observation restricting the search space in which key candidates with probabilities below $p^*$ may be found.

**Theorem 1.** *Let $d$ non-increasing subkey probability distributions $P_i$ for $1 \leq i \leq d$ be given and let the correct key be $k^* = (k_1, ..., k_d)$. Let $p^* = P_1[k_1] \cdot ... \cdot P_d[k_d]$ be the probability of the correct key. Then, it holds that all the keys $(x_1, ..., x_d)$ s.t. $P_1[x_1] \cdot ... \cdot P_d[x_d] \geq p^*$ have subkeys in the range $1 \leq x_i \leq n_i^*$ where $n_i^* = max_{1 \leq l \leq N}\{l : P_i(l) \cdot \prod_{j \neq i} P_j(1) \geq p^*\}$.*

**Proof:** For any dimension $i$, if we choose the most likely value in all dimensions $j \neq i$, we can find the minimum probability in dimension $i$ that still fulfills the condition $P_1[x_1] \cdot ... \cdot P_d[x_d] \geq p^*$. Therefore, any index $l$ s.t. $P_i(l)$ is smaller than this minimum probability will never be part of a key whose probability is higher than or equal to $p^*$. Since $P_j$ is non-increasing, the most likely value is at index 1, hence for all dimensions $j \neq i$, the most likely value has probability $P_j(1)$. We look for the farthest index $l$ in dimension $i$ such that the product $P_i(l) \cdot \prod_{j \neq i} P_j(1)$ is still higher than or equal to $p^*$. Therefore, the number of subkey indexes for dimension $i$ that are needed in order to compute the rank of a key whose probability $p^*$ is

$$n_i^* = max_{1 \leq l \leq N}\{l : P_i(l) \cdot \prod_{j \neq i} P_j(1) \geq p^*\}. \qquad \square$$

This bound $n_i^*$ can be easily computed in $O(\log N)$ time using a binary search. Theorem 1 gives us an upper bound on the rank of the correct key, that is tighter than the trivial bound of [23]:

**Corollary 1.** *(Box Upper Bound) Given $d$ non-increasing subkey probability distributions $P_i$ for $1 \leq i \leq d$ then, the rank of the correct key $k^* = (k_1, ..., k_d)$ whose probability $p^*$ is*

$$Rank(k^*) \leq \prod_{i=1}^{d} n_i^* - \prod_{i=1}^{d} (n_i^* - k_i).$$

### 3.2 Upper Bound Functions

As we shall see in Section 4, our general idea is to upper bound each subkey's probability distribution $P_i$ by an integrable function $f_i$, such that $P_i[j] \leq f_i(j)$ for all $1 \leq j \leq N$. Then, given the correct key $k^* = (k_1, ..., k_d)$ and using analytical methods, we can estimate the rank of $k^*$ by a closed formula. To do this, let us first define $Rank_f(k^*)$:

**Definition 2.** *A function is an upper bound function $f$ of a sorted probability distribution $P$ of size $N$ if for all $j = 1, ..., N$, $P[j] \leq f(j)$.*

**Definition 3 ($Rank_f(k^*)$).** *Given $d$ non-increasing subkey probability distributions $P_i$ and their corresponding $d$ upper bound functions $f_i$ s.t. $P_i[j] \leq f_i(j)$ for all $1 \leq i \leq d$ and $1 \leq j \leq N$, and let the correct key be $k^* = (k_1, ..., k_d)$. Let $p^* = P_1[k_1] \cdot ... \cdot P_d[k_d]$ be the probability of the correct key. Then, define $Rank_f(k^*)$ to be the number of keys $(x_1, ..., x_d)$ s.t. $f_1(x_1) \cdot ... \cdot f_d(x_d) \geq p^*$.*

**Proposition 1.** *Given $d$ subkey probability distributions $P_i$ and their $d$ upper-bound functions $f_i$ and given the correct key $k^* = (k_1, ..., k_d)$ and its probability $p^* = P_1[k_1] \cdot ... \cdot P_d[k_d]$, it holds that $Rank(k^*) \leq Rank_f(k^*)$*

**Proof:** Let the sets $R$ and $R_f$ be defined as follows:

$$R = \{(x_1, .., x_d) \mid \prod_{i=1}^{d} P_i[x_i] \geq p^*\}$$

$$R_f = \{(x_1, .., x_d) \mid \prod_{i=1}^{d} f_i(x_i) \geq p^*\}$$

For each $(x_1, ..., x_d) \in R$, it holds that $\prod_{i=1}^{d} P_i[x_i] \geq p^*$. Since $f_i(x_i) \geq P_i[x_i]$ for each $i \in [1, .., d]$, it holds

$$\prod_{i=1}^{d} f_i(x_i) \geq \prod_{i=1}^{d} P_i[x_i]$$

therefore

$$\prod_{i=1}^{d} f_i(x_i) \geq p^*$$

and therefore $(x_1, ..., x_d) \in R_f$. □

Our idea is to calculate $Rank_f(k^*)$ of the correct key $k^*$ whose probability is $p^*$, by integrating the volume under the manifold derived by the $d$ upper bound functions $f_i$, subject to the isotropic curve defined by $\prod_{i=1}^{d} f_i(x_i) \geq p^*$:

$$Rank(k^*) \leq Rank_f(k^*) \leq \int_{\substack{0 \leq x_1, ..., x_d \leq N, \\ \prod_{i=1}^{d} f_i(x_i) \geq p^*}} \cdots \int 1 \, dx_1 \ldots dx_d. \tag{1}$$

### 3.3 Pareto-like Functions

Following Section 2, we decided to choose the integrable upper-bound functions to be Pareto distributions. However, for an upper bound function we do not need to have an actual probability distribution. Therefore we focus on Pareto-like functions, without the requirement that $\int f(x)dx = 1$ and without the requirement that $\alpha > 1$.

**Definition 4.** *A function $f(x)$ is Pareto-like if $f(x) = a/x^\alpha$ for some $a > 0$ and $\alpha \geq 0$.*

Given a probability distribution $P$ we find it useful to consider *anchored* Pareto-like functions, that are defined by two indexes $l < r$, as follows.

**Definition 5.** *Let $P$ be a distribution, and let indexes $1 \leq l < r \leq N$ be given. Then a function $f(x)$ is anchored at $l, r$ if $f(l) = P[l]$ and $f(r) = P[r]$. We call the indexes $l, r$ the* anchors *of $f$.*

**Lemma 1.** *Let $P$ be a distribution, and let indexes $1 \leq l < r \leq N$ be given. Let $\alpha = \log_{r/l}(P[l]/P[r])$ and let $a = P[r]r^\alpha$, or equivalently, $a = P[l]l^\alpha$. Then $f(x) = a/x^\alpha$ is the unique Pareto-like function that is anchored at $l, r$.*

### 3.4 The Existence of Pareto-like Upper Bound Functions

**Proposition 2.** *Given a sorted non-increasing probability distribution $P$, there exists an index $r > 1$ such that the Pareto-like function $f(x)$ that is anchored at $1, r$ is an upper bound function.*

**Proof:** Let $f^{\alpha,a} = a/x^\alpha$ be a Pareto-like function with parameters $\alpha$ and $a$. Since $P$ is a sorted non-increasing probability distribution, $P[1]$ is greater than or equal to any other $P[j]$ for $1 \leq j \leq N$. Therefore, a trivial Pareto-like upper bound function for $P$ is $f^{0,P[1]}(x) = P[1]$. If $P[2] = P[1]$ then $f^{0,P[1]}(x)$ fulfills the requirements with anchors 1,2.

Else, we shall construct a upper-bound function $f^{\alpha,P[1]}$, that is anchored at $1, r$ for some $r > 1$. Given any index $r > 1$ and the value $P[r]$, let $\alpha_r = \log_r(P[1]/P[r])$. Then, the function $f^{\alpha_r,P[1]}(x) = P[1]/x^{\alpha_r}$ is the unique Pareto-like function that is anchored at $1, r$. Hence we need to find $1 < r \leq N$ s.t. $f^{\alpha_r,P[1]}(x)$ is an upper-bound function: $f^{\alpha_r,P[1]}(x) \geq P[x]$ for all $1 \leq x \leq N$. For a fixed index $r > 1$, the function $g(\alpha) = f^{\alpha,P[1]}(r) = P[1]/r^\alpha$ for $\alpha \geq 0$ is monotone decreasing, with $g(0) = P[1]$. Clearly $g(\alpha_r) = P[r]$, hence for all $0 \leq \alpha \leq \alpha_r$ we have $f^{\alpha,P[1]}(r) \geq P[r]$. Let $\alpha^* = \min_r\{\alpha_r\}$ and let $r^* = \arg\min_r\{\alpha_r\}$ be the minimal index at which $\alpha^*$ is achieved. Then, $f^{\alpha^*,P[1]}$ is an upper bound function for $P$ which obeys $f^{\alpha^*,P[1]}(1) = P[1]$ and $f^{\alpha^*,P[1]}(r^*) = P[r^*]$ - at all other indices $r \neq 1$, $r \neq r^*$ we have $\alpha^* \leq \alpha_r$ by definition, so $f^{\alpha^*,P[1]}(r) = g(\alpha^*) \geq g(\alpha_r) = P[r]$. □

### 3.5 Efficient Search for Pareto-like Upper Bound functions.

In this section we prove a complete characterization of all the different Pareto-like upper bound functions that are anchored to indexes of the distribution $P$. This characterization is described by the following theorem:

**Theorem 2.** *Given a non-increasing sorted probability distribution $P$, there exist $m < N$ indexes $t_1, \ldots, t_m$ such that every unique Pareto-like upper bound function for $P$ that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for some $1 \leq j < m$.*

**Proof:** We prove this theorem using induction. The base case is Proposition 2, which shows that $t_1 = 1$, and whose proof describes how to find $t_2 = r$.

For the induction step we assume that we have a Pareto-like upper bound function $f$ that is anchored at indexes $l, r$. We prove that if there exists another Pareto-like upper bound function $\hat{f} \neq f$ that is anchored at $\hat{l}, \hat{r}$ s.t. $\hat{l} > l$ then there exists some $r < t \leq \hat{l}$ s.t. the Pareto-like function anchored at $r, t$ is an upper-bound function for $P$.

We shall prove this in 3 steps: in Theorem 3 we prove that the anchors of $f$ and $\hat{f}$ cannot be nested. In Theorem 4 we prove that if the anchors of $f$ and $\hat{f}$ are interleaved then the intermediate anchors coincide, i.e., $t = r = \hat{l}$. Finally in Theorem 5 we prove that if the anchors of $f$ and $\hat{f}$ obey $r < \hat{l}$ then the required $t$ exists and obeys $t \leq \hat{l}$.

To prove these theorems, we first state two simple lemmas, and prove Propositions 3 and 4 showing that two different Pareto-like upper bound functions can "share" only their left anchors $l$, or only their right anchor $r$, but not both.

**Lemma 2.** *Let $f_1 = a/x^\alpha$ and $f_2 = b/x^\beta$ be Pareto-like functions s.t. $\alpha > \beta$. Then, $f_1(x) = f_2(x)$ only at the crossover point $x_c = (a/b)^{\alpha-\beta}$.*

**Lemma 3.** *Let $f_1 = a/x^\alpha$ and $f_2 = b/x^\beta$ be Pareto-like functions s.t. $\alpha > \beta$ and let $x_c$ be their crossover point. Then for $x > x_c$ $f_1 < f_2$ and for $x < x_c$ $f_2 < f_1$.*

**Proposition 3.** *Given a non-increasing sorted probability distribution $P$ and a Pareto-like upper bound function $f$ of $P$ anchored at $l, r$, then any Pareto-like function $\hat{f} \neq f$ that is anchored at $l, \hat{r}$ s.t. $\hat{r} > r$ will violate the upper bound condition at index $r$, i.e., $\hat{f}(r) < P[r]$.*

**Proof:** Let $f(x) = a/x^\alpha$ and $\hat{f}(x) = \hat{a}/x^{\hat{\alpha}}$ be defined as above, "sharing" the anchor $l$. Since $f$ is a Pareto-like upper bound function $f(\hat{r}) \geq P[\hat{r}]$. By definition $P[\hat{r}] = \hat{f}(\hat{r})$, therefore $f(\hat{r}) \geq \hat{f}(\hat{r})$, which is equivalent to

$$a \cdot \hat{r}^{\hat{\alpha}} \geq \hat{a} \cdot \hat{r}^{\alpha}.$$

Substituting $a = P[l] \cdot l^\alpha$ and $\hat{a} = P[l] \cdot l^{\hat{\alpha}}$ for the same $l$ we get

$$\hat{r}^{\hat{\alpha}-\alpha} \geq l^{\hat{\alpha}-\alpha}.$$

Since $\hat{r} \geq l$, we get $\hat{\alpha} > \alpha$. Since $P[r] = f(r)$ by definition, in order to prove $P[r] > \hat{f}(r)$, it suffices to prove $f(r) > \hat{f}(r)$, which is equivalent to

$$\hat{a} \cdot r^{\alpha} < a \cdot r^{\hat{\alpha}}.$$

By substituting $a = P[l] \cdot l^{\alpha}$ and $\hat{a} = P[l] \cdot l^{\hat{\alpha}}$ we get the equivalent inequality

$$l^{\hat{\alpha}-\alpha} < r^{\hat{\alpha}-\alpha},$$

which holds since $l < r$ and $\hat{\alpha} > \alpha$. □

**Proposition 4.** *Given a non-increasing sorted probability distribution $P$ and a Pareto-like upper bound function $f$ of $P$ anchored at $l, r$, then any Pareto-like function $\hat{f} \neq f$ that is anchored at $\hat{l}, r$ for some $\hat{l} < l$ will violate the upper bound condition at index $l$, i.e., $\hat{f}(l) < P[l]$.*

**Proof:** Analogous to that of Proposition 3.

**Theorem 3.** *(No nested anchors). Let $P$ be a non-increasing sorted probability distribution and let two index pairs $l < r$ and $\hat{l} < \hat{r}$ s.t. $l \leq \hat{l} < \hat{r} \leq r$. There cannot exist two Pareto-like upper bound functions $f \neq \hat{f}$ s.t. $f$ is anchored at $l, r$ and $\hat{f}$ is anchored at $\hat{l}, \hat{r}$.*

**Proof:** From Proposition 3, we see that there cannot exist two different Pareto-like upper bound functions $f, \hat{f}$ s.t. $f(l) = P[l] = \hat{f}(l)$ (i.e., the crossover point is at $l$) but $f(r) = P[r]$ and $\hat{f}(\hat{r}) = P[\hat{r}]$ for $r < \hat{r}$. In the same way, from Proposition 4 we see that there cannot exist two different Pareto-like upper bound functions $f, \hat{f}$ s.t. $f(r) = P[r] = \hat{f}(r)$ (i.e., the crossover point is at $r$) but $f(l) = P[l]$ and $\hat{f}(\hat{l}) = P[\hat{l}]$ for $l < \hat{l}$. Therefore the only option is that $l < \hat{l} < \hat{r} < r$. However, this option also cannot exist since then we get

$$\begin{aligned} \hat{f}(l) \geq P[l] = f(l), \quad f(\hat{l}) \geq P[\hat{l}] = \hat{f}(\hat{l}), \\ f(\hat{r}) \geq P[\hat{r}] = \hat{f}(\hat{r}), \quad \hat{f}(r) \geq P[r] = f(r). \end{aligned}$$

However, since $f \neq \hat{f}$ at least 3 of these 4 inequalities must be sharp, which means we need to have at least two crossover points, contrary to Lemma 2. □

**Theorem 4.** *(Interleaved anchors). Let $P$ be a non-increasing sorted probability distribution and let two index pairs $l < r$ and $\hat{l} < \hat{r}$ be such that $l \leq \hat{l} \leq r \leq \hat{r}$ and s.t. there exist two different Pareto-like upper bound functions, $f$ anchored at $l, r$ and $\hat{f}$ anchored at $\hat{l}, \hat{r}$. Then, $l < \hat{l} = r < \hat{r}$.*

**Proof:** From properties of $f$ and $\hat{f}$, it holds that:

$$\begin{aligned} \hat{f}(l) \geq P[l] = f(l), \quad f(\hat{l}) \geq P[\hat{l}] = \hat{f}(\hat{l}), \\ \hat{f}(r) \geq P[r] = f(r), \quad f(\hat{r}) \geq P[\hat{r}] = \hat{f}(\hat{r}). \end{aligned} \tag{2}$$

In other words, we get $f(l) \leq \hat{f}(l)$, $f(\hat{l}) \geq \hat{f}(\hat{l})$, $f(r) \leq \hat{f}(r)$ and $f(\hat{r}) \geq \hat{f}(\hat{r})$. Notice that $f$ and $\hat{f}$ are different, therefore they have a single crossover point. To obtain a contradiction, assume that $l < \hat{l} < r < \hat{r}$. If two or more of the inequalities in Equation (2) are equalities then $f \equiv \hat{f}$, contrary to the premise. Therefore at least 3 of the inequalities in Equation (2) are sharp. However since $f \neq \hat{f}$ there is a unique crossover point $x_c$ between them, and regardless of where $x_c$ is located with respect to $l, \hat{l}, r, \hat{r}$, there will be two indices $u, v \in \{l, \hat{l}, r, \hat{r}\}$ either to its left or to its right, such that $f(u) > \hat{f}(u)$ and $f(v) < \hat{f}(v)$, contradicting Lemma 2. Therefore at least two indices need to be equal to each other. From Proposition 3 we have that there cannot exist two upper bound Pareto-like functions $f$ and $\hat{f}$ such that $f(l) = P[l] = \hat{f}(l)$ (i.e., the crossover point is at $l$) but $f(r) = P[r]$ and $\hat{f}(\hat{r}) = P[\hat{r}]$ for $r < \hat{r}$. Therefore $l$ cannot be equal to $\hat{l}$. In the same way, Proposition 4 shows that there cannot exist two Pareto-like upper bound functions $f$ and $\hat{f}$ such that $f(r) = P[r] = \hat{f}(r)$ (i.e., the crossover point is at $r$) but $f(l) = P[l]$ and $\hat{f}(\hat{l}) = P[\hat{l}]$ for $l < \hat{l}$. Therefore, $r$ cannot be equal to $\hat{r}$. Therefore, the only option is $\hat{l} = r$. □

**Theorem 5.** *(Disjoint anchors). Let $P$ be a non-increasing sorted probability distribution and let two index pairs $l < r$ and $\hat{l} < \hat{r}$ be such that $l < r \leq \hat{l} < \hat{r}$ and s.t. there exist two different Pareto-like upper bound functions, $f$ anchored at $l, r$ and $\hat{f}$ anchored at $\hat{l}, \hat{r}$. Then there exists $r < t \leq \hat{l}$ such that the Pareto-like function $\bar{f}$ anchored at $r, t$ is an upper-bound function.*

**Proof:** We prove the theorem by constructing a series of candidate functions until we find one that meets the requirements. The first candidate function $f_1$ is the Pareto-like function that is anchored at $r, r+1$. Since $f(r) = f_1(r)$ and $f(r+1) \geq f_1(r+1)$ it holds that $f_1(i) \geq P(i)$ for each $i \leq r+1$. Therefore we need to check whether $f_1(i) \geq P(i)$ for each $i > r+1$. We start from $i = r+2$ and increase $i$ until we find the first $i$ that violates $f_1$, i.e., $f_1(i) < P[i]$. If no such $i$ is found then $f_1$ is an upper bound and the proof is done. If such $i$ is found, we change the candidate Pareto-like function to be $f_2$ that is anchored at $r, i$. Notice that candidate $f_2$ is an upper bound for each $j \leq i$, since $f_1$ is an upper bound for each $j \leq i$ and $f_2(j) \geq f_1(j)$ for each $j \in [r, i]$ and $f_2(j) > f(j)$ for each $j \leq r$. Therefore we need to check whether $f_2$ is an upper bound for $j > i$. We continue in the same way increasing $i$ and looking for violations till $i \leq \hat{l}$. For each $j \geq \hat{l}$ it holds that the current candidate Pareto-like $f_c$ that anchored at $r, i$ is an upper bound since $f_c(i) \leq \hat{f}(i)$ and $f_c(\hat{l}) \geq \hat{f}(\hat{l})$. □

The combination of Theorems 3, 4 and 5 completes the proof of Theorem 2. □

From Theorem 2 we get an efficient search method (Algorithm 1) to find all the different Pareto-like upper-bound functions.

For a given $l$, the algorithm finds its leftmost matching $r > l$ so the Pareto-like function anchored at $l, r$ is an upper bound function, and then for the next candidate pair it sets $l = r$. Theorem 2 guarantees that no valid candidate pairs are missed by this skip. To do this, the algorithm starts with $l = 1$ and $r = 2$. In

**Algorithm 1:** The function ParetoUpperEstimation.

**Input:** Subkey distributions $P$.
**Output:** A set $C$ of candidate pairs.

```
1  C = ∅; l = 1;
2  while (l < N) do
3  |  r = l + 1;
4  |  found = False;
5  |  while (r ≤ N and found == False) do
6  |  |  α = log_{r/l}(P[l]/P[r]);
7  |  |  a = P[l] · l^α;
8  |  |  k = r;
9  |  |  while k < N do
10 |  |  |  d = a/k^α − P[k];
11 |  |  |  if d ≥ 0 then
12 |  |  |  |  k = min(N, ⌊(a/P[k])^{1/α}⌋) + 1 ; // f(k) ≥ P[k]: jump forward
13 |  |  |  if d < 0 or k == N then
14 |  |  |  |  break;
15 |  |  if d < 0 then
16 |  |  |  r = k;              // violation of upper bound: switch to (l, k)
17 |  |  else
18 |  |  |  found = True;
19 |  |  |  C = C ∪ {(l, r)};
20 |  |  |  l = r;
21 return C;
```

order to check whether this pair defines a Pareto-like upper bound the algorithm iterates over all $k \geq r$.

Rather than test all values of $k \geq r$, in order to speed up the calculation, if $f_{l,r}(k) \geq P[k]$, i.e., $d > 0$ (line 11), the algorithm "jumps forward". It calculates the intersection point between the candidate Pareto function and $P[k]$, i.e., the $k'$ such that $a/k'^{\alpha} = P[k]$ (line 12) and "jumps" to this $k'$. The reason this jump is valid is as follows: For each $h \in [k, k'], P[k] \geq P[h] \geq P[k']$ and for each $t \in [k, k'], a/k^{\alpha} \geq a/t^{\alpha} \geq a/k'^{\alpha}$. Since $a/k'^{\alpha} = P[k]$ it holds $a/t^{\alpha} \geq P[h]$, for each $t \in [k, k']$ and $h \in [k, k']$. Therefore $f_{l,k}$ is guaranteed to be an upper bound for each $h \in [k, k']$ and we only need to check whether it is an upper bound beyond $k'$, therefore $k$ is updated to be $k'$ (line 12).

If $d < 0$ (line 15), the algorithm finds the first violation, i.e., $P[k] > f_{l,r}(k)$, therefore, it stops on this $k$. Since $k$ is a violation index, all the pairs $(l, t)$ such that $t < k$ do not anchor Pareto-like upper bounds: clearly any such $(l, t)$ anchors a Pareto-like function with a violation at $k$. Therefore, the next candidate for the leftmost matching $r > l$ is $k$ and we only need to check whether $f_{l,k}$ is also an upper bound for the indices $t > k$. So the algorithm sets $r$ to $k$ (line 16) and repeats till it finds a Pareto-like upper bound. Then, the algorithm sets the $l$ of the next pair to be $r$, according to Theorem 2, and continues in the same way.

**Algorithm 2:** PickBest: Choosing the best Pareto-like upper bounds.

**Input:** $\{C_i, P_i, k_i\}_{i=1}^{d}$ s.t. $C_i$ is a set of candidate pairs for Subkey distribution $P_i$, and the correct key $k^* = \{k_i\}_{i=1}^{d}$

**Output:** $(\{a_i\}_{i=1}^{d}, \{\alpha_i\}_{i=1}^{d})$

1 **for** $i = 1$ *to* $d$ **do**

2 $\quad (l_i, r_i) = \arg\min_{(l,r)\in C_i^{10}} \{|P_i[k_i] - f_{l,r}(k_i)|\}$;

3 $\quad \alpha_i = \log_{r_i/l_i}(P_i[l_i]/P_i[r_i])$;

4 $\quad a_i = P_i[l_i] \cdot l_i^{\alpha}$;

5 return $\{a_i\}_{i=1}^{d}, \{\alpha_i\}_{i=1}^{d}$;

**Proposition 5.** *Let $P$ be a non-increasing sorted probability distribution and let $m$ be the number of the its anchors as in Theorem 2. The running time of Algorithm 1 is $O(m \cdot N)$.*

**Proof:** To find the first pair $(t_1 = 1, t_2)$ that anchors a Pareto-like upper bound takes $N-1$ steps. Then, starting at $t_2$, looking for its leftmost matching $t_3 > t_2$, the algorithm tests for violations at indices between $t_2$ and $N$, taking at most $N - t_2$ steps and similarly till finding last pair takes $N - t_m$ steps. In total we get at most $(N-t_1)+(N-t_2)+...+(N-t_m) = m \cdot N - \sum_{i=1}^{m} t_i = O(m \cdot N)$. □

Note that since typically $m \ll N$ the algorithm is almost linear in $N$ and very quick in practice. Furthermore, while the "forward jumps" in the algorithm do not affect the asymptotic running time, they have a dramatic impact in practice since they often allow skipping hundreds of candidates per jump.

### 3.6 Choosing the best Pareto-like upper bound function

In general, Algorithm 1 identifies multiple candidates for Pareto-like upper bound functions for each distribution $P_i$. We need to select the 'best' function per distribution in the sense that it will lead to a tight bound in the volume computed in Equation (1). To do so, we need to select the criteria for the 'best' Pareto-like upper bound function.

We tested many criteria for selecting the upper-bound functions. Overall we found that there is no clear 'best' upper bound function for a given probability distribution: rather, the best bound usually depends on the rank $k_i$ of the correct subkey value, with larger ranks $k_i$ requiring upper-bound functions anchored at larger indices. After much experimentation we arrived at the following choice: Given the indices of the correct key $k^* = (k_1, k_2, ..., k_d)$, for each $P_i$ we choose the pair $(l_i, r_i)$ which anchors a Pareto-like upper bound function $f_{l,r}$ such that $f(k_i)$ will be the closest to $P_i[k_i]$. Since larger $k_i$ require larger indices $(l_i, r_i)$ which provide larger $a_i$ which directly influences on the upper bound (as we shall see in section 4.1), we limit the chosen pair to be one of the first $w$ pairs. Note that the choice of $w$ impacts the running time (a smaller $w$ means fewer options to minimize over in Algorithm 2 line 2) and potentially the accuracy of the resulting bounds. In our experiments we tested values $w \in [5, 50]$ and found

**Algorithm 3:** Pareto Rank Estimation.

**Input:** Subkey distributions $\{P_i\}_{i=1}^{d}$, the correct key $k^* = \{k_i\}_{i=1}^{d}$
**Output:** Upper bound rank of the correct key.

Let $p^* = \prod_{i=1}^{d} P_i[k_i]$;
**for** *i=1 to d* **do**
  $C_i = ParetoUpperEstimation(P_i)$;
$\{a_i\}_{i=1}^{d}, \{\alpha_i\}_{i=1}^{d} = PickBest(\{C_i\}_{i=1}^{d}, \{P_i\}_{i=1}^{d}, k^*)$;
return UpperBound$(\{a_i\}_{i=1}^{d}, \{\alpha_i\}_{i=1}^{d}, \{P_i\}_{i=1}^{d}, p^*)$;

that in fact the bounds were quite insensitive to the choice of $w$. Therefore we selected $w = 10$ arbitrarily. We denote the set of the first 10 pairs of $C_i$ by $C_i^{10}$. Algorithm 2 shows the pseudo code for the selection method.

Note that instead of first building the whole set $C_i$ as in Algorithm 1, we can build $C_i$ incrementally until we find the 'best' pair.

## 4 PRank: The Pareto Rank Estimation Algorithm

Now that we know how to efficiently obtain Pareto-like upper bound functions for all the subkey distributions, we describe the details of our rank estimation algorithm (See Algorithm 3). First, we upper bound each one of the $d$ probability distributions by a Pareto-like upper bound function. Then, given the probability of the correct key, we compute the upper bound rank by a closed formula.

### 4.1 Estimating the Volume

We solve the multiple integral Equation (1) for the Pareto-like upper bound functions $f_i$, for the general case $d \geq 2$. We assume a general configuration in which $\alpha_i \neq \alpha_j$ for all $i \neq j$.

We solve the multiple integral Equation (1) for the Pareto-like upper bound functions $f_i$, for the first trivial case which is $d = 2$, and we get:

$$\left(\frac{a_1 \cdot a_2}{p^*}\right)^{\frac{1}{\alpha_1}} \cdot \frac{\alpha_1}{\alpha_1 - \alpha_2} \cdot N^{\frac{\alpha_1 - \alpha_2}{\alpha_1}} + \left(\frac{a_1 \cdot a_2}{p^*}\right)^{\frac{1}{\alpha_2}} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_1} \cdot N^{\frac{\alpha_2 - \alpha_1}{\alpha_2}}.$$

Now, in more details, we solve the multiple integral Equation (1) for the Pareto-like upper bound functions $f_i$, for the first non-trivial case which is $d = 3$. Then, we will generalize it for any $d$. Plugging the Pareto-like function $f_i = a_i / x_i^{\alpha_i}$ for each $1 \leq i \leq d$ into Equation (1) we get:

$$\int_0^N \int_0^N \int_0^N \underset{\left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{x_3^{\alpha_3}}\right) \geq p^*}{1} \, dx_3 dx_2 dx_1. \tag{3}$$

We assume the general case in which $\alpha_i \neq \alpha_j$ for all $i \neq j$. The range of the multiple integrals in Equation (3) is equivalent to

$$\left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_3}} \geq x_3.$$

Therefore, we can plug $x_3$ into Equation (3) to get

$$\int_0^N \int_0^N \left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_3}} dx_2 dx_1. \tag{4}$$

The lower bound of each dimension $x_i$ is 0, but Pareto-like functions are not defined at 0. However, in order to maintain $x_3 \leq N$ we require:

$$x_3 \leq \left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_3}} \leq N,$$

which provides a bound on $x_2$

$$x_2 \geq \left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{N^{\alpha_3}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_2}}. \tag{5}$$

We denote this lower bound on $x_2$ by $x_2'$:

$$x_2' \triangleq \left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{N^{\alpha_3}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_2}}.$$

For all $x_2 < x_2'$, we get $x_3 > N$ which is out of range, therefore for $x_2 < x_2'$ we take $x_3 = N$. By splitting the inner-most integral in Equation (4) into two ranges we get:

$$\int_0^N \left[\int_0^{x_2'} N\, dx_2 + \int_{x_2'}^N \left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_3}} dx_2\right] dx_1. \tag{6}$$

We repeat the procedure and divide the next dimension $x_1$. In order to maintain $x_2 \leq N$, from Equation (5)

$$\left(\frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{N^{\alpha_3}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_2}} \leq x_2 \leq N,$$

so $x_1$ should maintain

$$x_1 \geq \left(\frac{a_1}{N^{\alpha_2}} \cdot \frac{a_2}{N^{\alpha_3}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_1}}.$$

Denote this lower bound of $x_1$ by $x_1'$

$$x_1' \triangleq \left(\frac{a_1}{N^{\alpha_2}} \cdot \frac{a_2}{N^{\alpha_3}} \cdot \frac{a_3}{p^*}\right)^{\frac{1}{\alpha_1}}.$$

For all $x_1 < x_1'$, we get $x_2 > N$ which is out of range, therefore for $x_1 < x_1'$ we take $x_2 = N$. Plugging this into Equation (6)

$$\int_0^{x_1'} \int_0^N N\, dx_1 + \int_{x_1'}^N \left[ \int_0^{x_2'} N\, dx_2 + \int_{x_2'}^N \left( \frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*} \right)^{\frac{1}{\alpha_3}} dx_2 \right] dx_1. \quad (7)$$

To solve this integral we solve each term separately, starting we the inner-right term of Equation (7):

$$\int_{x_2'}^N \left( \frac{a_1}{x_1^{\alpha_1}} \cdot \frac{a_2}{x_2^{\alpha_2}} \cdot \frac{a_3}{p^*} \right)^{\frac{1}{\alpha_3}} dx_2,$$

which is straightforward:

$$\left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_3}} \int_{x_2'}^N x_2^{-\frac{\alpha_2}{\alpha_3}}\, dx_2 = \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot x_2^{1-\frac{\alpha_2}{\alpha_3}} \Bigg|_{x_2'}^N.$$

After substituting the limits, since $\alpha_2 \neq \alpha_3$, we get:

$$\left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot \left[ N^{\frac{\alpha_3 - \alpha_2}{\alpha_3}} - \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_2} - \frac{1}{\alpha_3}} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}} \right]$$

which equals to

$$\left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot N^{\frac{\alpha_3 - \alpha_2}{\alpha_3}} - \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_2}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}}. \quad (8)$$

Now, we calculate the inner-left term of Equation (7):

$$\int_0^{x_2'} N\, dx_2 = \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_2}} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}}. \quad (9)$$

Plugging in Equations (8) and (9) into the right term of Equation (7), we get:

$$\int_{x_1'}^N \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot N^{\frac{\alpha_3 - \alpha_2}{\alpha_3}} + \left( \frac{a_1 \cdot a_2 \cdot a_3}{x_1^{\alpha_1} \cdot p^*} \right)^{\frac{1}{\alpha_2}} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_3} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}}\, dx_1.$$

Calculating the integral we get:

$$\left( \frac{a_1 \cdot a_2 \cdot a_3}{p^*} \right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot N^{\frac{\alpha_3 - \alpha_2}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_1} \cdot x_1^{\frac{\alpha_3 - \alpha_1}{\alpha_3}} \Bigg|_{x_1'}^N +$$
$$\left( \frac{a_1 \cdot a_2 \cdot a_3}{p^*} \right)^{\frac{1}{\alpha_2}} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_3} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_1} \cdot x_1^{\frac{\alpha_2 - \alpha_1}{\alpha_2}} \Bigg|_{x_1'}^N.$$

Solving this and adding the solution of the left term of Equation (7) we get the final formula for $d = 3$:

$$\begin{aligned}
&\left(\frac{a_1 \cdot a_2 \cdot a_3}{p^*}\right)^{\frac{1}{\alpha_1}} \cdot \frac{\alpha_1}{\alpha_1 - \alpha_2} \cdot \frac{\alpha_1}{\alpha_1 - \alpha_3} \cdot N^{\frac{\alpha_1 - \alpha_2}{\alpha_1}} \cdot N^{\frac{\alpha_1 - \alpha_3}{\alpha_1}} + \\
&\left(\frac{a_1 \cdot a_2 \cdot a_3}{p^*}\right)^{\frac{1}{\alpha_2}} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_3} \cdot \frac{\alpha_2}{\alpha_2 - \alpha_1} \cdot N^{\frac{\alpha_2 - \alpha_1}{\alpha_1}} \cdot N^{\frac{\alpha_2 - \alpha_3}{\alpha_2}} + \\
&\left(\frac{a_1 \cdot a_2 \cdot a_3}{p^*}\right)^{\frac{1}{\alpha_3}} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_1} \cdot \frac{\alpha_3}{\alpha_3 - \alpha_2} \cdot N^{\frac{\alpha_3 - \alpha_1}{\alpha_3}} \cdot N^{\frac{\alpha_3 - \alpha_2}{\alpha_3}}
\end{aligned} \tag{10}$$

For the general case $d \geq 2$ the analysis is analogous so we omit the details. The final solution is the following closed formula:

$$\sum_{i=1}^{d} \left[ \left( \frac{1}{p^*} \cdot \prod_{j=1}^{d} a_j \right)^{\frac{1}{\alpha_i}} \cdot \prod_{j=1, j \neq i}^{d} \left( \frac{\alpha_i}{\alpha_i - \alpha_j} \cdot N^{\frac{\alpha_i - \alpha_j}{\alpha_i}} \right) \right]. \tag{11}$$

The same analysis can also be done assuming each dimension has a different bound—and then we can use the $n_i^*$ of the Box Bound (Theorem 1) to yield a closed formula for the upper bound:

$$Rank(k^*) \leq \sum_{i=1}^{d} \left[ \left( \frac{1}{p^*} \cdot \prod_{j=1}^{d} a_j \right)^{\frac{1}{\alpha_i}} \cdot \prod_{j=1, j \neq i}^{d} \left( \frac{\alpha_i}{\alpha_i - \alpha_j} \cdot n_j^{*\frac{\alpha_i - \alpha_j}{\alpha_i}} \right) \right]. \tag{12}$$

Notes: (i) The formulas of Equations (12) and (11) are analogous to the results of [5] obtained via Laplace transforms. (ii) In our data we did not encounter cases in which $\alpha_i = \alpha_j$ so the "general configuration" assumption did not restrict us. (iii) Equations (12) and (11) sum $d$ terms with alternating signs and large absolute values which can cause numerical stability challenges. To address this, we sum them in decreasing order of absolute values. Further, in order to improve the stability of the computation, rather than multiply values in the inner product, we take the log of the expression, sum the logs, and exponentiate back.

### 4.2 Theoretical Worst-case Performance

**Running Time:** Equation (12) consists of $d$ additions and in each sum we have $d$ multiplications and $d$ calls to the real-value power function. Therefore, assuming that calculating $x^y$ takes constant time, the running time of computing the formula is $O(d^2)$. According to Proposition 5 the running time of finding a Pareto-like upper bound function $f_i$ for each probability distribution $P_i$, takes $O(m_i \cdot N)$. Let $\hat{m} = \max_i\{m_i\}$, then the running time in total is $O(\hat{m} \cdot d \cdot N)$. Since typically $\hat{m} \ll N$ the algorithm is almost linear in $d \cdot N$ and very quick in practice.

**Space Complexity:** The algorithm needs to keep for each probability distribution its corresponding Pareto-like upper bound function. In other words, it only needs to keep the corresponding $a_i$, $\alpha_i$ and $n_i^*$ for every $1 \leq i \leq d$. Therefore the space complexity is $O(d)$.

**Algorithm 4:** UpperBound.

**Input:** Subkey distributions $\{P_i\}_{i=1}^{d}$, Pareto distributions $\{a_i\}_{i=1}^{d}$, $\{\alpha_i\}_{i=1}^{d}$ and the correct key probability $p$.

**Output:** Upper bound for the rank of the correct key.

1 **for** *i=1 to d* **do**

2 $\quad n_i = \max_{1 \le l \le N}\{l : P_i(l) \cdot \prod_{j \neq i} P_j(1) \ge p\}$;

3 $ub = \sum_{i=1}^{d} \left[ \left( \frac{1}{p} \cdot \prod_{j=1}^{d} a_j \right)^{\frac{1}{\alpha_i}} \cdot \prod_{j=1, j \neq i}^{d} \left( \frac{\alpha_i}{\alpha_i - \alpha_j} \cdot n_i^{\frac{\alpha_i - \alpha_j}{\alpha_i}} \right) \right]$;

4 return ub;

## 5 Performance Evaluation

We evaluated the performance of the PRank estimation algorithm through an extensive simulation study. We compared the new PRank algorithm to the currently best rank estimation algorithm: the histogram algorithm of [13]. We implemented both in Matlab. We ran both algorithms on a 2.80GHz i7 PC with 8GB RAM running Microsoft windows 7, 64bit.

### 5.1 Data Corpus I

To evaluate PRank, we used the data of [11]. Within this data, there are 611 probability distribution sets gathered from a specific SCA. The SCA of [11] was against AES [1] with 128-bits keys. The sets represent various setting of the SCA: number of traces used, whether the clock was jittered, and the values of tunable attack parameters. The attack grouped the key bits into 16 8-bit subkeys, and hence its output probability distributions are over these byte values. Each set in the corpus consists of the correct secret key and 16 distributions, one per subkey. The distributions are sorted in non-increasing order of probability, each of length $2^8$.

Since we don't know the real rank of the correct keys, we used the histogram rank as the $x$ axis in our resulting graphs (Figures 2 and 3). We measured the time and the upper bound for each trace using PRank and the histograms rank estimation.

We checked PRank's accuracy and running time for different configurations. We started with $d = 16$ and $n = 2^8$. As we shall see, the computed upper bound is noticeably higher than the histogram rank, however the running time is a fraction of that of the histogram algorithm.

Next, in order to improve the accuracy, we applied a technique suggested in [13]: merge the $d = 16$ probability lists of size $n = 2^8$ into $d = 8$ lists of size $n = 2^{16}$. As we shall see we found that reducing the number of dimensions indeed significantly improved the accuracy with a marginal increase in the PRank running time.

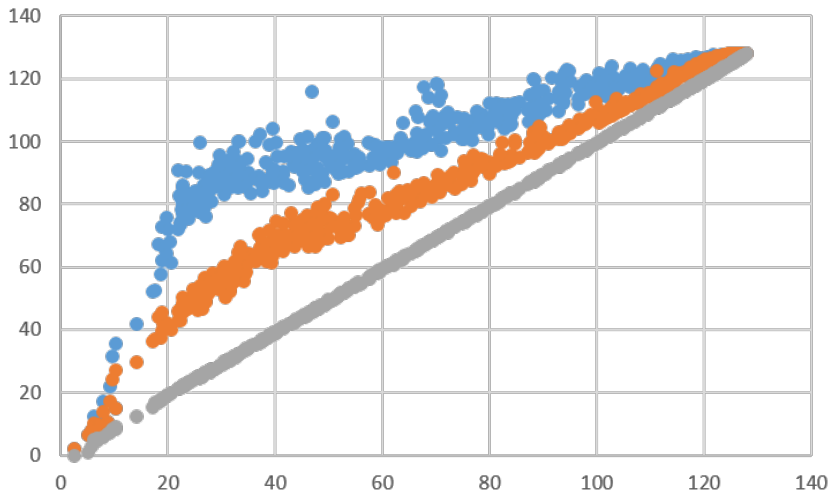


**Fig. 2.** Ranks ($\log_2$) as a function of histogram rank. The curves are, from top to bottom: PRank upper bound for $d = 16$ (blue), PRank upper bound for $d = 8$ (orange) and Histogram rank for B=5K (gray).

**Bound Tightness** Figure 2 illustrates the PRank upper bound with $d = 16$, the PRank upper bound with $d = 8$ and the histogram rank, all in the number of bits ($\log_2$), as function of the number of bits of the histogram rank. The figure clearly shows that it is advantageous to reduce the dimension $d$. As we can see in the Figure, the accuracy of PRank's estimation is quite good: for ranks between $2^{80}$–$2^{100}$ the median PRank bound is less than 10 bits above the histogram rank, and for the very high ranks (above $2^{100}$) median PRank bound is only 4 bits more. For small ranks, around $2^{20}$, PRank gave a bound which is roughly 20 bits greater than that of the histogram—however we argue that such ranks are within reach of key enumeration so rank estimation is not particularly interesting there.

**Runtime Analysis** Figure 3 shows the running times (in seconds) of the histogram rank estimation (with B=5K and B=50K) and the PRank estimation for $d = 16$ and $d = 8$. The running time of the PRank consists of the preprocessing time of finding the Pareto-like upper bound function of each probability distribution, plus the running time of calculating the closed formula bound of Equation (12) given the secret key. The histogram running time consists of the preprocessing of converting each probability distribution into a histogram plus the running time of finding the sum of the corresponding bins given the secret key. The figure shows that PRank, in both $d = 8$ and $d = 16$, runs faster than the Histograms in its 4 configurations. Looking at the PRank itself we can see, as we expected, that $d = 16$ runs faster than $d = 8$ since the length $N$ of each distribution is shorter. Looking at the Histograms runtimes, we can see that

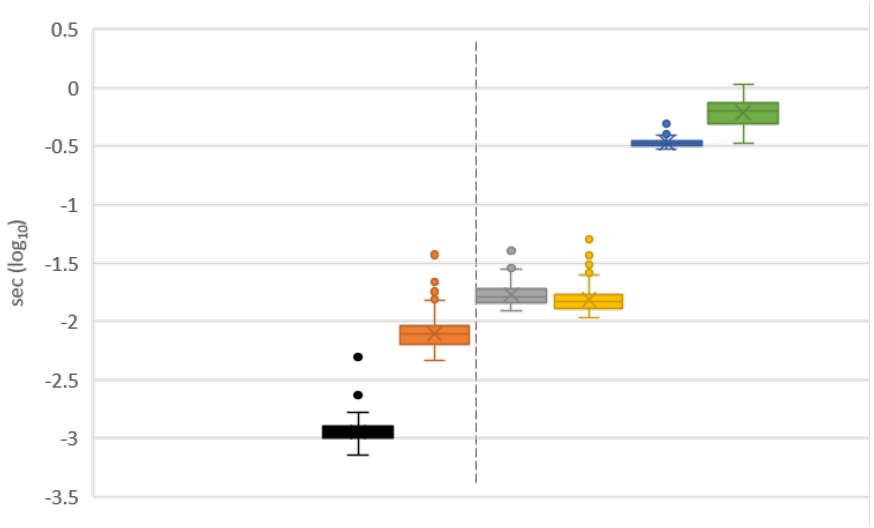


**Fig. 3.** The running times of the algorithms in seconds (log-scale) as a box plot: the top and bottom of each box represent the $3^{rd}$ and $1^{st}$ quartiles, respectively, and the line inside the box represents the median. The boxes to the left of the dashed line represent the PRank running time in two configurations: $d = 16$ (black) and $d = 8$ (orange). The boxes on the right represent the histograms running time in four configurations: $d = 16$, B=5K (gray), $d = 8$, B=5K (yellow), $d = 16$, B=50K (blue) and $d = 16$, B=50K (green), all in seconds (log scale).

B=5K is faster than B=50K and for B=50K we can also see that $d = 16$ is faster than $d = 8$. However, notice that both PRank and Histograms run in less than 1 second.

**Space Utilization** Table 1 illustrates the space used by the 2 algorithms' data structures. As we can see, the memory consumption of PRank algorithm is drastically lower than the histogram space consumption. PRank space consumption is trivial $3 \cdot d$ while the histogram space requirements are around $2 \cdot B \cdot d$.

### 5.2 Data Corpus 2

The second set of experiments uses the data of [17] described in Section 2. Each set in the corpus consists of the correct secret key and $d = 4$ distributions, one per sub-key. The distributions are sorted in non-increasing order of probability. Each distribution has at most $2^{17}$ subkeys values, hence the maximal full-key rank that could be predicted by them is at most $2^{68}$.

We again used the histogram rank as the $x$ axis in our resulting graphs. We measured the time, the space consumption and the upper bound for each trace using PRank and the histograms rank estimation.

| | B=5K | | B=50K | |
|---|---|---|---|---|
| $d = 8$ | 24 bytes | 80KB | 24 bytes | 800KB |
| $d = 16$ | 48 bytes | 160KB | 48 bytes | 1.6MB |

**Table 1.** Space complexity of PRank and Histograms in four configurations. The left side of each column in the table is the PRank space and the right is the Histograms space.

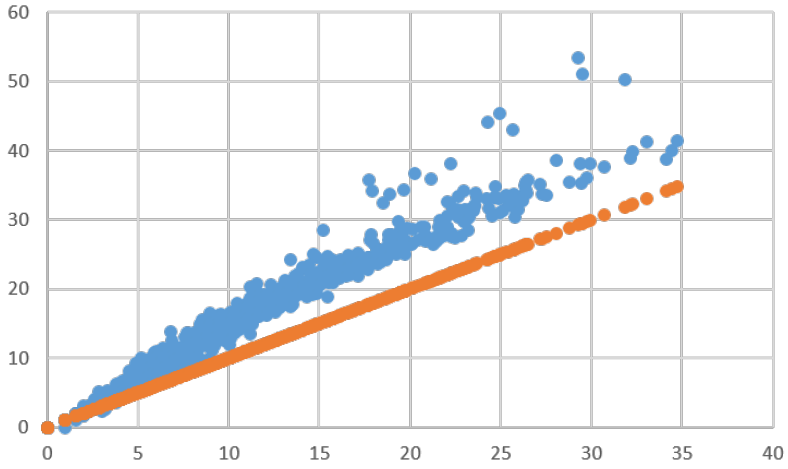


**Fig. 4.** Ranks ($\log_2$) as a function of histogram rank. The curves are, from top to bottom: PRank upper bound for $d = 4$ (blue) and Histogram rank (orange).

**Bound Tightness** Figure 4 illustrates the number of bits ($\log_2$) of the PRank upper bound and the histogram rank. All these values are shown as function of the number of bits of histogram rank, hence its curve is a straight line. The figure clearly shows that it is advantageous to reduce the dimension $d$ comparing to the $d = 8$ and $d = 16$ of the Data corpus I. As we can see in Figure 4, the average difference between the PRank and Histogram ranks is 3.97 bits.

**Runtime Analysis** Figure 5 shows the ($\log_{10}$) time (in seconds) of histogram rank estimation [13] and PRank estimation.

The figure shows that on this data corpus too the PRank algorithm is faster than the histogram algorithm—by about 1 order of magnitude—but both algorithms are very efficient, taking under 1 second to complete.

## 6 Conclusion

In this paper we proposed a new method called PRank for rank estimation, that is conceptually simple, and more time and memory efficient than previous

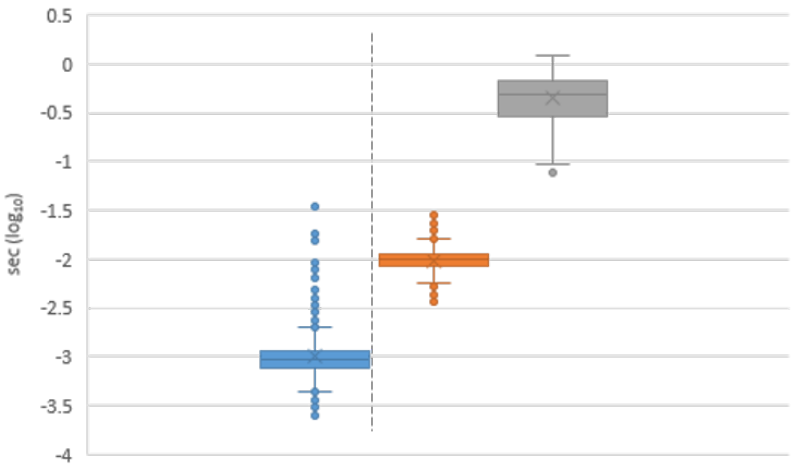


**Fig. 5.** The left side represents the PRank running time for $d = 4$ (blue). The right side represents the histograms running time in two configurations: $d = 4$, B=5K (orange) and $d = 4$, B=50K (gray), all in seconds (log scale).

proposals. Our main idea is to bound each subkey distribution by a Pareto-like function: since these are analytical functions, we can then estimate the rank by a closed formula.

We started by investigating the prediction quality of SCA-based probability distribution functions on the data corpus of [17]. We showed that the SCA probability is overly optimistic; the predicted Guessing Entropy is much lower than the average true rank. Moreover, the true rank distribution has a long tail: large ranks do appear with non-negligible probability. We showed that a Pareto distribution is a good model for the true rank distribution.

Motivated by this observation, we used Pareto-like functions to upper-bound the empirical SCA-based probability distributions. We fully characterized such upper-bounding functions and developed an efficient algorithm to find them. We then used Pareto-like functions to develop a new explicit upper bound on the rank of a given key. Combined with the algorithm to find the upper-bounding Pareto-like functions, we obtained an $O(dN^2)$ rank upper-bound estimation algorithm we call PRank. We also developed a simple and efficient local-search rank lower-bound algorithm.

We evaluated the performance of PRank through extensive simulations based on two real SCA data corpus, and compared it to the currently-best histogram-based algorithm. We showed that PRank gives a good rank estimation with much improved time and memory efficiency, especially for large ranks: For ranks between $2^{80} - 2^{100}$ PRank estimation is at most 10 bits above the histogram rank and for ranks beyond $2^{100}$ the PRank estimation is only 4 bits above the histogram rank—yet it runs faster, and uses negligible memory. PRank gives

a new and interesting method to solve the rank estimation problem based on reduction to analytical functions and calculating one closed formula hence using negligible time and space. It is therefore a useful addition to the SCA evaluator's toolbox.

## References


1. FIPS PUB 197, advanced encryption standard (AES), 2001. U.S. Department of Commerce/National Institute of Standards and Technology (NIST).
2. D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The EM side-channel(s). In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 29–45. 2003.
3. Barry C Arnold. *Pareto distribution*. Wiley Online Library, 1985.
4. D.J. Bernstein, T. Lange, and C. van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. Cryptology ePrint Archive, Report 2015/221, 2015. `http://eprint.iacr.org/`.
5. Markus Bibinger. Notes on the sum and maximum of independent exponentially distributed random variables with different scale parameters. *arXiv preprint arXiv:1307.3945*, 2013.
6. Andrey Bogdanov, Ilya Kizhvatov, Kamran Manzoor, Elmar Tischhauser, and Marc Witteman. Fast and memory-efficient key recovery in side-channel attacks. In *Selected Areas in Cryptography (SAC)*, 2015.
7. Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 13–28. 2003.
8. Marios O Choudary and PG Popescu. Back to massey: Impressively fast, scalable and tight security evaluation tools. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 367–386. Springer, 2017.
9. L. David and A. Wool. A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In *Proc. RSA Conference Cryptographers Track (CT-RSA'17), LNCS 10159*, pages 311–327, San Francisco, February 2017. Springer Verlag.
10. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 401–429. Springer, 2015.
11. D. Fledel and A. Wool. Sliding-window correlation attacks against encryption devices with an unstable clock. Cryptology ePrint Archive, Report 2018/317, 2018. `https://eprint.iacr.org/2018/317`.
12. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems—CHES 2001*, pages 251–261. Springer, 2001.
13. Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schueth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption*, pages 117–129, 2015.
14. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer, 1999.
15. Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113, 1996.
16. Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In *Advances in Cryptology–ASIACRYPT 2015*, pages 313–337. Springer, 2015.

17. Yossef Oren, Ofir Weisse, and Avishai Wool. A new framework for constraint-based probabilistic template side channel attacks. In *Cryptographic Hardware and Embedded Systems–CHES 2014*, pages 17–34. Springer, 2014.
18. Jing Pan, Jasper GJ Van Woudenberg, Jerry I Den Hartog, and Marc F Witteman. Improving DPA by peak distribution analysis. In *Selected Areas in Cryptography (SAC)*, pages 241–261. Springer, 2011.
19. Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: an integrated approach. In *Proc. 18th Cryptographic Hardware and Embedded Systems–CHES 2016*, pages 61–81. Springer, 2016.
20. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001.
21. François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology-EUROCRYPT 2009*, pages 443–461. Springer, 2009.
22. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *International Conference on Selected Areas in Cryptography*, pages 390–406. Springer, 2012.
23. Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In *Advances in Cryptology–EUROCRYPT 2013*, pages 126–141. Springer, 2013.
24. Xin Ye, Thomas Eisenbarth, and William Martin. Bounded, yet sufficient? how to determine whether limited side channel information enables key recovery. In *Smart Card Research and Advanced Applications (CARDIS)*, pages 215–232. 2014.