Rank Estimation with Bounded Error via Exponential Sampling

Liron David
School of Electrical Engineering
Tel Aviv University
Ramat Aviv, 69978, Israel
lirondavid@gmail.com

Avishai Wool
School of Electrical Engineering
Tel Aviv University
Ramat Aviv, 69978, Israel
yash@eng.tau.ac.il

Abstract—Efficient rank estimation algorithms are of prime interest in security evaluation against side-channel attacks (SCA) and recently also for password strength estimators. In a side channel setting 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). In password strength estimators the rank estimation allows estimating how many attempts a password cracker would need until it finds a given password.

We propose ESrank, the first rank estimation algorithm with a bounded error ratio: its error ratio is bounded by γ^{2d-2}, for any probability distribution, where d is the number of subkey dimensions and $\gamma > 1$ can be chosen according to the desired accuracy. ESrank is also the first rank estimation algorithm that enjoys provable poly-logarithmic time- and space-complexity. Our main idea is to use exponential sampling to drastically reduce the algorithm's complexity.

We evaluated the performance of ESrank on real SCA and password strength corpora. We show ESrank gives excellent rank estimation with roughly a 1-bit margin between lower and upper bounds in less than 1 second on the SCA corpus and 4 seconds preprocessing time and 7μsec lookup time on the password strength corpus.

Index Terms—Side channel, Password Strength Estimation, Rank Estimation, Key Enumeration[1]

I. Introduction

A. 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 [17], power consumption [16], electromagnetic emulation [28], electromagnetic radiation [2][13] 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) using "reasonable" efforts.

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 [24], [30], [8]. 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 by 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 (in the optimal order) without finding the correct key, and then conclude that the implementation is practically secure 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.

[1] A preliminary version of this paper appeared in the proceeding of the CT-RSA 2019 conference.

In addition to the side channel scenario, rank estimation is useful for password strength estimation. Text passwords are currently the most popular authentication method and are still in widespread use. Unfortunately, users often choose predictable and easy passwords, enabling password guessing attacks. Password strength estimators are used to help users avoid picking weak passwords. The most precise definition of password's strength is the number of attempts that an attacker would need in order to guess it [10]. Recently it was shown in [9] that password strength estimation can be cast in the framework of the rank estimation with d "dimension" such as prefix, suffix, shift-pattern and more.

Formally, we define the rank estimation [31] problem as follows: 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. 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 number of dimensions d, or k^*'s rank, are small, one can easily compute the rank of the correct key by a straightforward key enumeration. However, for a key with a high rank r, any optimal-order key enumeration requires $\Omega(r)$ time — which may be prohibitive, and the currently-best optimal-order key enumeration algorithm [30] requires $\Omega(N^{d/2})$ space, which again may be prohibitive. Hence developing fast and low-memory algorithms to estimate the rank, without enumeration, is of great interest.

B. Related work

The best key enumeration algorithm so far, in terms of optimal-order, was presented by Veyrat-Charvillon, Gérard, Renauld and Standaert in [30]. 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, all assuming independent per-dimension probability distributions [6], [23], [27], [33], [8], [20], [18], [19], [22], [29]. 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: enumerating an exponential key space will always come at an exponential cost. 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. [31]. 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 in 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, progressively refining 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 [31]. In particular, Glowacz et al. [14] presented a rank estimation algorithm that is based on a convolution of histograms and allows estimating the key rank of (even high rank) keys. A comparable result was developed independently by Bernstein et al. [5]. This Histogram algorithm is currently the best rank estimation algorithm we are aware of. The space complexity of this algorithm is $O(dB)$ where d is the number of dimensions and B is a design parameter controlling the number of the histogram bins. However, as we shall see, the Histogram algorithm's error ratio is not bounded – it depends on the probability distributions.

And, since its accuracy depends on the probability distribution, the desired accuracy cannot be chosen from advance.

Martin et al. [23] 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. Subsequently, in [21] Martin et al. show that their algorithm [23] is mathematically equivalent to the Histogram algorithm [14] for a suitable choice of their respective discretization parameter, thus they can both be equally accurate. Since the two algorithms are equivalent we compared our algorithm's performance only to that of the Histogram algorithm [14].

Ye et al. investigated an alternative solution based on a weak Maximum Likelihood (wML) approach [33], 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. [11] 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. Recently, Wang at al. [32] presented a rank estimation for at dependent score lists.

Choudary et al. [7] 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, the actual guessing entropy (GE), i.e., the rank of the correct key, is sometimes quite different from the expectation. In contrast, our algorithm focuses on the real GE.

Poussier et al. [26] compare several solution according to their maximum or weak maximum likelihood strategy. Grosso [15] presents a rank estimation with a trade-off between the efficiency and the tightness that is suitable for large keys. In addition, using backtracking, a parallel key enumeration is obtained.

Rank estimation is also useful for password strength estimations. Recently [9] presented the first method for estimating the strength of human-chosen text passwords that is able to tweak the estimation according to each user's personal context, without retraining its model of rank estimation. The idea is to cast the question in the probabilistic framework. Each password is viewed as a point in a d-dimensional search space with dimensions such as the base word, prefix, suffix, capitalization, l33t pattern etc. The probability distribution of each dimension is learned separately. This learning process is based on empirical password frequencies extracted from leaked password corpora, that are projected onto the d dimensions. Once the d probability distributions are learned, the a-priori probability of a given password is the product of the d probabilities of its sub-passwords. Using this model, optimal-order password cracking is done by searching the space in decreasing order of a-priori password probability, which is analogous to side-channel key enumeration; likewise, password strength estimation is analogous to side-channel rank estimation.

C. Contributions

In this paper we propose a simple and effective new rank estimation method called ESrank, that is fundamentally different from previous approaches. The ESrank algorithm is the first rank estimation algorithm that enjoys provable poly-logarithmic time- and space-complexity, and which has bounded error ratio of γ^{2d-2}, for any probability distributions, where d is the number of dimensions and $\gamma > 1$ can be chosen according to the desired accuracy. Our main idea is to use exponential sampling to drastically reduce the algorithm's complexity.

We rigorously analyze its accuracy, time and space complexities. We prove ESrank has a poly-logarithmic time- and space-complexity: for a design parameter $1 < \gamma \leq 2$ ESrank has time complexity of $O(d^{2+\epsilon}(\log_\gamma N)^{2+\epsilon})$ for $0 < \epsilon < 1$ and space complexity of $O(d(\log_\gamma N)^2)$, and it can be driven to any desired level of accuracy (trading off time and space against accuracy).

We then compared ESrank to the currently-best histogram based algorithm. We show that the Histogram's accuracy depends on the probability distribution, and that there exist distributions for which its estimation error is unbounded.

After rigorously bounding the accuracy, time and space complexities, we evaluated the performance of ESrank on a real SCA data corpus and on a password strength corpus, and compared it to the histogram-based algorithm. We show that ESrank gives excellent rank estimation (with roughly a 1-bit margin between lower and upper bounds), with a performance that is on-par with the Histogram algorithm: a run-time of under 1 second on a standard laptop. On the password strength corpus, ESrank achieves 1-bit accuracy with 4 seconds prerocessing time, 7μsec lookup time. On the same corpus we find that Histogram's estimation error cannot be made to vanish for any choice of parameters.

Our implementation is available as open-source from [3].

II. The ESrank Algorithm for the case $d = 2$

We start with describing the idea of our algorithm in case $d = 2$, then we shall extend this idea for the general case $d \geq 2$.

A. An exact rank estimation for $d = 2$

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. $P_1[x_1] \cdot ... \cdot P_d[x_d] \geq p^*$.

Note that we require each probability distribution P_i to be non-increasing (rather than strictly decreasing) is since it may contain equal probabilities: in fact observing equal-probability values is extremely common in real corpora.

Definition 2: Let 2 non-increasing subkey probability distributions P_1 and P_2, each of size N, the correct key $k^* = (k_1, k_2)$ and an index $1 \leq i \leq N$ be given. Let $p^* = P_1[k_1] \cdot P_2[k_2]$ be the probability of the correct key. Then define H_i to be the number of points (i, j) such that $P_1[i] \cdot P_2[j] \geq p^*$, i.e.,

$$H_i(k^*) = |\{(i, j) | P_1[i] \cdot P_2[j] \geq p^*\}|.$$

The idea of the algorithm is to find $H_i(k^*)$ for each i. The rank of the correct key k^* is the sum of $H_i(k^*)$ over $1 \leq i \leq N$, i.e.,

$$Rank(k^*) = \sum_{i=1}^{N} H_i(k^*).$$

The pseudo code is described in Algorithm 1. The correctness of Algorithm 1 stems from the observation that $H_i \geq H_{i+1}$ for all $1 \leq i \leq N - 1$. Therefore, to find H_{i+1}, j starts from H_i and it is decreased until H_{i+1} is found.

Proposition 1: The running time of Algorithm 1 is $\Theta(N)$.
Proof: At the beginning $i = 1$ and $j = N$. In each iteration either i is increased by 1 or j is decreased by 1 until either

Algorithm 1 Exact Rank algorithm.

Input: Two non-increasing probability distributions P_1, P_2 of size N each, the correct key $k^* = (k_1, k_2)$ and its probability $p^* = P_1[k_1] \cdot P_2[k_2]$.
Output: $Rank(k^*)$.

```
1: i = 1; j = N; rank = 0
2: while i ≤ N and j ≥ 1 do
3:    p = P1[i] · P2[j]
4:    if p ≥ p* then
5:       rank = rank + j  {j is H_i(k*)}
6:       i = i + 1
7:    else
8:       j = j − 1
9:    end if
10: end while
11: return rank
```

$i = N + 1$ or $j = 0$. Therefore, the number of steps is at most $2 \cdot N$ in case both i and j reach their limits, and is at least N in case only one of them reaches it limit. Therefore, the running time is $\Theta(N)$. □

Algorithm 1 is reminiscent of the Threshold key enumeration algorithm of [20].

B. Exponential Sampling with $d = 2$

To make this algorithm sub-linear, we use exponential sampling. Intuitively, we sample a set of indices SI and run Algorithm 1 on the $SI \times SI$ grid. On the sampled indices Algorithm 1 is no longer exact, but we can modify it to produce lower and upper bounds on $Rank(k^*)$. As we shall see, if we use exponential sampling, we can bound the inaccuracy introduced by the sampling.

Given a non-increasing subkey probability distribution P of size N, the exponential sampling process returns a sampled probability distribution (SI, SP) of size N_s where $N_s = O(\log N)$. SI contains the sampled indices and SP contains their corresponding probabilities such that $SP[i] = P[SI[i]]$ for all $i \leq N_s$.

The goal of the exponential sampling is to maintain an invariant on the ratio between sampled indices. Let $1 < \gamma \leq 2$ be given and let b be the smallest i such that $i/(i-1) \leq \gamma$. The first b sampled indices are the first b indices of P. The rest of the sampled indices are sampled from P at powers of γ. Formally,

Definition 3: Given a non-increasing subkey probability distribution P of size N, the exponential sampling process returns a sampled probability distribution (SI, SP) of size N_s such that for all $i \leq N_s - 1$:

$$\begin{cases} SI[i] = i & \text{if } i \leq b \\ \\ SI[i]/SI[i-1] \leq \gamma \text{ and} \\ SI[i+1]/SI[i-1] > \gamma & \text{otherwise.} \end{cases} \quad (1)$$

E.g., if $\gamma = 2$ then $b = 2$, and for $SI = \{1, 2, 4, 8, \ldots, N\}$ invariant (1) holds. The pseudo code of this sampling is described in Algorithm 2. Note that the indices 1 and N are always included in SI.

Algorithm 2 Exponential Sampling Process.

Input: A probability distribution P of size N, b, γ.
Output: A sampled probability distribution (SI, SP).

```
1: for i = 1 to b do
2:    SI[i] = i; SP[i] = P[i]
3: end for
4: j = b; i = j + 1; c = j + 1
5: while i < N do
6:    if i/j ≤ γ and (i + 1)/j > γ then
7:       SI[c] = i; SP[c] = P[i]
8:       c = c + 1; j = i
9:    end if
10:   i = i + 1
11: end while
12: SI[c] = N; SP[c] = P[N]
13: return (SI, SP)
```

Lemma 1: If SI is the output of Algorithm 2 then for any index $i \geq b + 1$ in SI it holds that

$$SI[i] = \lfloor \gamma \cdot SI[i-1] \rfloor.$$

and

$$SI[i] - SI[i-1] = \lfloor (\gamma - 1) \cdot SI[i-1] \rfloor.$$

Proof: According to Algorithm 2, it holds that

$$\frac{SI[i]}{SI[i-1]} \leq \gamma \quad \text{and} \quad \frac{SI[i]+1}{SI[i-1]} > \gamma.$$

Therefore,

$$\gamma \cdot SI[i-1] - 1 < SI[i] \leq \gamma \cdot SI[i-1].$$

Since $SI[i]$ is an integer it holds

$$SI[i] = \lfloor \gamma \cdot SI[i-1] \rfloor.$$

The difference $SI[i] - SI[i-1]$ obeys

$$(\gamma-1) \cdot SI[i-1] - 1 < SI[i] - SI[i-1] \leq (\gamma-1) \cdot SI[i-1].$$

The indices of SI are integers, therefore we get

$$SI[i] - SI[i-1] = \lfloor (\gamma - 1) \cdot SI[i-1] \rfloor.$$

□

Proposition 2: Let $N_s = |SI|$ be the size the sample returned by Algorithm 2. Then

$$b + \log_\gamma(N/b) \leq N_s < b + \log_\gamma(N/(b-1)).$$

Proof: Since $b \cdot \gamma^{Ns} \geq N$ and $b \cdot \gamma^{Ns-1} < N$.

To calculate the upper and lower bounds of the correct key $k^* = (k_1, k_2)$ given two sampled probability distributions, we generalize $H_i(k^*)$ for the sampled case:

Definition 4: Let a sampled probability distribution (SI, SP_1) of size N_s, a probability distribution P_2 of size N, the correct key $k^* = (k_1, k_2)$, its probability p^*

and an index $1 \leq i \leq N_s$ be given. Then define H_i^S to be the number of points (i,j) s.t. $1 \leq j \leq N$ and $SP_1[i] \cdot P_2[j] \geq p^*$, i.e.,

$$H_i^S(k^*) = |\{(i,j) | 1 \leq j \leq N \text{ and } SP_1[i] \cdot P_2[j] \geq p^*\}|.$$

The difference between every two successive indices in the sampled probability distributions might be bigger than 1, i.e., $SI[i+1] - SI[i] > 1$ therefore, besides counting H_i^S for each $i \leq N_s$ we also need to add the number of points (i', j) such that $SI[i] < i' < SI[i+1]$ for each $i \leq N_s - 1$. Recall that Algorithm 2 always includes $i = N$ in SI.

Definition 5: Let a sampled probability distribution (SI, SP_1) of size N_s, a probability distribution P_2 of size N, the correct key $k^* = (k_1, k_2)$, its probability p^* and an index $1 \leq i \leq N_s$ be given. Then define $H_{a,b}^S$ be the number of (i,j) s.t. $1 \leq j \leq N$ and $SI[a] < i < SI[b]$ and $SP_1[i] \cdot P_2[j] \geq p^*$, i.e.,

$$H_{a,b}^S(k^*) = |\{(i,j) | 1 \leq j \leq N \text{ and } SP_1[i] \cdot P_2[j] \geq p^* \text{ and } SI[a] < i < SI[b]\}|.$$

The idea of Algorithm 3 is to find $H_i^S(k^*)$ for each $i \in \{1, ..., N_s\}$ and $H_{i,i+1}^S(k^*)$ for each $i \in \{1, ..., N_s - 1\}$. The rank of the correct key k^* is the following sum:

$$Rank(k^*) = \sum_{i=1}^{N_s} H_i^S(k^*) + \sum_{i=1}^{N_s - 1} H_{i,i+1}^S(k^*).$$

Since we are given sampled distributions, we cannot calculate the exact values of $H_i^S(k^*)$ and $H_{i,i+1}^S(k^*)$. Instead we calculate upper and lower bounds for each $H_i^S(k^*)$ and $H_{i,i+1}^S(k^*)$ as illustrated in Figure 1.

Definition 6: Let $up(H_i^S(k^*))$ be an upper bound of $H_i^S(k^*)$ and let $up(H_{i,i+1}^S(k^*))$ be an upper bound of $H_{i,i+1}^S(k^*)$, i.e.,

$$H_i^S(k^*) \leq up(H_i^S(k^*))$$

and

$$H_{i,i+1}^S(k^*) \leq up(H_{i,i+1}^S(k^*)).$$

Definition 7: Let $low(H_i^S(k^*))$ be a lower bound of $H_i^S(k^*)$ and let $low(H_{i,i+1}^S(k^*))$ be a lower bound of $H_{i,i+1}^S(k^*)$, i.e.,

$$H_i^S(k^*) \geq low(H_i^S(k^*))$$

and

$$H_{i,i+1}^S(k^*) \geq low(H_i^S(k^*)).$$

Therefore, it holds

$$\sum_{i=1}^{N_s} low(H_i(k^*)) + \sum_{i=1}^{N_s - 1} low(H_{i,i+1}(k^*)) \leq Rank(k^*) \leq \sum_{i=1}^{N_s} up(H_i(k^*)) + \sum_{i=1}^{N_s - 1} up(H_{i,i+1}(k^*)). \quad (2)$$

Our algorithm is intuitively similar to exponential searching [4]; note that in our case the parameter γ is fractional.

Algorithm 3 Calculating Upper and Lower bounds.

Input: Sampled probability distributions SP_1, SP_2 each of size N_s, b, the correct key $k^* = (k_1, k_2)$ probability p^*.

Output: Upper and lower bounds on $Rank(k^*)$.

```
1: i = 1; j = N_s; ub = 0; lb = 0; uPrev = 0
2: while i ≤ N_s and j ≥ 1 do
3:    pCurr = SP_1[i] · SP_2[j]
4:    if pCurr ≥ p* then
5:       u = l = SI[j];
6:       ub = ub + u; lb = lb + l;
7:       if i ≥ b + 1 then
8:          ub = ub + uPrev · (SI[i] − SI[i − 1] − 1)
9:          lb = lb + l · (SI[i] − SI[i − 1] − 1)
10:      end if
11:      i = i + 1; uPrev = u
12:   else if j > 1 then
13:      pNext = SP_1[i] · SP_2[j − 1]
14:      if pNext < p* < pCurr then
15:         u = SI[j] − 1; l = SI[j − 1];
16:         ub = ub + u; lb = lb + l
17:         if i ≥ b + 1 then
18:            ub = ub + uPrev · (SI[i] − SI[i − 1] − 1)
19:            lb = lb + l · (SI[i] − SI[i − 1] − 1)
20:         end if
21:         i = i + 1; uPrev = u
22:      else
23:         j = j − 1
24:      end if
25:   else
26:      j = j − 1
27:   end if
28: end while
29: if j < 1 and i ≤ N_s then
30:    ub = ub + uPrev · (SI[i] − SI[i − 1] − 1)
31: end if
32: return (lb, ub)
```

C. Bounding the Sampled Distributions

Given two probability distributions P_1 and P_2, each of size N, we first sample the indices using Algorithm 2. We get sampled probability distributions (SI, SP_1) and (SI, SP_2) each of size N_s when SI is the set of sampled indices and SP_1, SP_2 are the corresponding sampled probabilities. Given these sampled probability distributions, the next step is to calculate an upper bound and a lower bound for $Rank(k^*)$. This is done in Algorithm 3.

To do this, Algorithm 3 keeps two variables: ub for the upper bound and lb for the lower bound. At the beginning, both ub and lb are initialized to 0.

In the algorithms analysis we find it useful to use the following definition:

Definition 8: Given a key k^*, and given $1 \leq i \leq N_s$, let u_i be the value of u at iteration i in Algorithm 3 and let

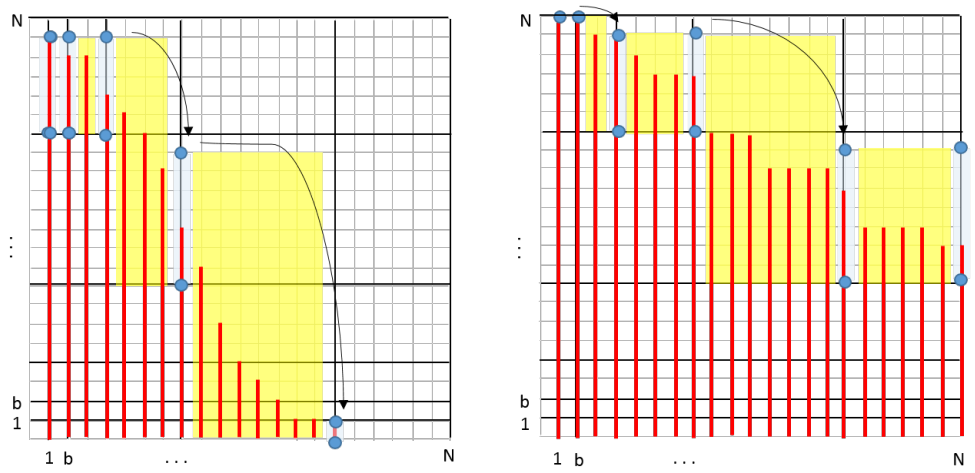

Fig. 1. The red bars represent the un-sampled H_i's, and the black grid represents the sampled indices in SI. For each sampled index $1 \le i \le N_s$ the blue circles are upper and lower bounds on $H_i^S(k^*)$. The yellow-shaded rectangles represent $H_{i,i+1}^S(k^*)$ for each $b \le i \le N_s - 1$, for two different keys. Note that the yellow-shaded rectangles stop exactly one index before the sampled indices, in both dimensions.

l_i be the value of l at iteration i in Algorithm 3.

Algorithm 3 starts with $i = 1$ and $j = N_s$. It decreases j until one of the two options happens:

(a) (line 14) We reach the highest j such that

$$SP_1[i] \cdot SP_2[j] < p^* < SP_1[i] \cdot SP_2[j-1].$$

In this case $(i, j-1) \in H_i^S(k^*)$ but $(i,j) \notin H_i^S(k^*)$, therefore

$$SI[j-1] \le H_i^S(k^*) \le SI[j] - 1.$$

Therefore the values of l_i and u_i become

$$l_i = SI[j-1] \text{ and } u_i = SI[j] - 1, \tag{3}$$

and the running totals ub and lb are updated (line 16).

(b) (line 4) We reach the highest j such that

$$SP_1[i] \cdot SP_2[j] \ge p^*.$$

In this case we have the exact value of $H_i^S(k^*)$ which is

$$H_i^S(k^*) = SI[j].$$

Therefore the values of l_i and u_i become

$$l_i = u_i = SI[j], \tag{4}$$

and the running totals ub and lb are updated (line 6).

In the next step, after finding bounds on H_i^S, the algorithm moves to $i+1$ and finds bounds on H_{i+1}^S. Since $H_i^S \ge H_{i+1}^S$ we start from j of the previous iteration i.e., j s.t. $SI[j-1] \le H_i^S \le SI[j]$ and decrease it to get the corresponding bounds on H_{i+1}^S.

Once $i \ge b+1$ (lines 7, 17) the difference $SI[i] - SI[i-1] \ge 1$ therefore $H_{i-1,i}^S(k^*) \ge 1$ and it should be added. To upper bound this number we multiply the upper bound of H_{i-1}^S, which is $uPrev_i = u_{i-1}$, by the width of $H_{i-1,i}^S(k^*)$, which is $(SI[i] - SI[i-1] - 1)$ (lines 8, 17). To lower bound $H_{i-1,i}^S(k^*)$ we multiply the lower bound of H_i^S, which is l_i by the width of $H_{i-1,i}^S(k^*)$, (lines 9, 18); see Figure 1.

Theorem 1: Let two sampled probability distributions SP_1 and SP_2, which are sampled from the probability distributions P_1 and P_2 respectively using Algorithm 2 with $\gamma > 1$ be given and let b be the smallest i such that $i/(i-1) \le \gamma$. For a key k^*, let ub and lb be the outputs of Algorithm 3. Then $ub/lb \le \gamma^2$.

Proof: From Equation (2) it holds that

$$ub = \sum_{i=1}^{N_s} up(H_i(k^*)) + \sum_{i=1}^{N_s-1} up(H_{i,i+1}(k^*)).$$

Since

$$up(H_i^S(k^*)) = u_i$$

and

$$up(H_{i,i+1}(k^*)) = u_i \cdot (SI[i+1] - SI[i] - 1)$$

we get

$$ub = \sum_{i=1}^{N_s} u_i + \sum_{i=1}^{N_s-1} u_i \cdot (SI[i+1] - SI[i] - 1).$$

Since $SI[i+1] - SI[i] = 1$ for all $1 \le i \le b-1$, the first $b-1$ elements of the second sum are 0.

$$\begin{aligned} ub = & \sum_{i=1}^{N_s} u_i + \sum_{i=b}^{N_s-1} u_i \cdot (SI[i+1] - SI[i]) - 1) \\ & \sum_{i=1}^{b-1} u_i + \sum_{i=b}^{N_s-1} \big(u_i + u_i \cdot (SI[i+1] - SI[i] - 1)\big) + u_{N_s} \\ & \sum_{i=1}^{b-1} u_i + \sum_{i=b}^{N_s-1} u_i \cdot (SI[i+1] - SI[i])) + u_{N_s} \end{aligned}$$

Separating the b'th term from the second sum we get

$$ub \leq \left(\sum_{i=1}^{b-1} u_i\right) + u_b \cdot (SI[b+1] - SI[b]) + u_{N_s} + \sum_{i=b+1}^{N_s-1} u_i \cdot (SI[i+1] - SI[i]). \quad (5)$$

Similarly from Equation (2) it holds that

$$lb = \sum_{i=1}^{N_s} low(H_i(k^*)) + \sum_{i=1}^{N_s-1} low(H_{i,i+1}(k^*)).$$

Since

$$low(H_i^S(k^*)) = l_i$$

and

$$low(H_{i,i+1}(k^*)) = l_{i+1} \cdot (SI[i+1] - SI[i] - 1)$$

(Note the shift in indices where the multiplication is by the lower bound of $i+1$) we get

$$lb = \sum_{i=1}^{N_s} l_i + \sum_{i=1}^{N_s-1} l_{i+1} \cdot (SI[i+1] - SI[i] - 1).$$

Again the first $b-1$ elements of the second sum are 0, therefore

$$lb = \sum_{i=1}^{N_s} l_i + \sum_{i=b}^{N_s-1} l_{i+1} \cdot (SI[i+1] - SI[i] - 1).$$

By shifting index i by 1 in the second sum, we get

$$\begin{aligned} lb &= \sum_{i=1}^{N_s} l_i + \sum_{i=b+1}^{N_s} l_i \cdot (SI[i] - SI[i-1] - 1) \\ &= \sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s} \left(l_i + l_i \cdot (SI[i] - SI[i-1] - 1)\right) \\ &= \sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s} l_i \cdot (SI[i] - SI[i-1]) \\ &\geq \sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s-1} l_i \cdot (SI[i] - SI[i-1]). \end{aligned} \quad (6)$$

In order to show $ub/lb \leq \gamma^2$, we prove the following two Lemmas:

Lemma 2:

$$\frac{\left(\sum_{i=b+1}^{N_s-1} u_i \cdot (SI[i+1] - SI[i])\right)}{\left(\sum_{i=b+1}^{N_s-1} l_i \cdot (SI[i] - SI[i-1])\right)} \leq \gamma^2 \quad (7)$$

Proof: We shall prove that for all $b+1 \leq i \leq N_s - 1$

$$u_i \cdot (SI[i+1] - SI[i])/l_i \cdot (SI[i] - SI[i-1]) \leq \gamma^2$$

and that will prove Equation (7). From Equation (3) and Equation (4) either $l_i = SI[j-1], u_i = SI[j] - 1$ or $l_i = u_i = SI[j]$, therefore

$$u_i/l_i \leq \gamma,$$

and we only need to prove

$$(SI[i+1] - SI[i])/(SI[i] - SI[i-1]) \leq \gamma. \quad (8)$$

From Lemma 1, it holds that

$$SI[i+1] - SI[i] = \lfloor (\gamma - 1) \cdot SI[i] \rfloor$$

and

$$SI[i] - SI[i-1] = \lfloor (\gamma - 1) \cdot SI[i-1] \rfloor$$

therefore Equation (8) is

$$\begin{aligned} \frac{SI[i+1] - SI[i]}{SI[i] - SI[i-1]} &= \frac{\lfloor (\gamma - 1) \cdot SI[i] \rfloor}{\lfloor (\gamma - 1) \cdot SI[i-1] \rfloor} \\ &\leq \frac{(\gamma - 1) \cdot SI[i]}{\lfloor (\gamma - 1) \cdot SI[i-1] \rfloor}. \end{aligned}$$

By Lemma 1 Equation (8) is

$$= \frac{(\gamma - 1) \cdot \lfloor \gamma \cdot SI[i-1] \rfloor}{\lfloor (\gamma - 1) \cdot SI[i-1] \rfloor} = \frac{(\gamma - 1) \cdot \lfloor \gamma \cdot SI[i-1] \rfloor}{\lfloor \gamma \cdot SI[i-1] - SI[i-1] \rfloor}.$$

Since $SI[i-1]$ is an integer it holds

$$\begin{aligned} &= \frac{(\gamma - 1) \cdot \lfloor \gamma \cdot SI[i-1] \rfloor}{\lfloor \gamma \cdot SI[i-1] \rfloor - \lfloor SI[i-1] \rfloor} \\ &= \gamma - 1 + \frac{(\gamma - 1) \cdot \lfloor SI[i-1] \rfloor}{\lfloor \gamma \cdot SI[i-1] \rfloor - \lfloor SI[i-1] \rfloor} \\ &= \gamma - 1 + \frac{(\gamma - 1) \cdot SI[i-1]}{\lfloor \gamma \cdot SI[i-1] - SI[i-1] \rfloor} \\ &= \gamma - 1 + \frac{(\gamma - 1) \cdot SI[i-1]}{\lfloor (\gamma - 1) \cdot SI[i-1] \rfloor} \leq \gamma. \end{aligned}$$

□

Lemma 3:

$$\frac{\left(\left(\sum_{i=1}^{b-1} u_i\right) + u_b \cdot (SI[b+1] - SI[b]) + u_{N_s}\right)}{\left(\sum_{i=1}^{b} l_i\right)} \leq \gamma^2. \quad (9)$$

Proof: We can write Equation 9 in the following way:

$$\frac{\left(\sum_{i=1}^{b-1} u_i\right) + u_b \cdot (SI[b+1] - SI[b]) + u_{N_s}}{\sum_{i=1}^{b} l_i} = \frac{\sum_{i=1}^{b} u_i}{\sum_{i=1}^{b} l_i} + \frac{u_b \cdot (SI[b+1] - SI[b] - 1) + u_{N_s}}{\sum_{i=1}^{b} l_i}.$$

Since $u_i/l_i \leq \gamma$, Equation (9) is upper bounded by

$$\leq \gamma + \frac{u_b \cdot (SI[b+1] - SI[b] - 1) + u_{N_s}}{\sum_{i=1}^{b} l_i}.$$

By Lemma 1 and since $l_1 \geq l_2 \geq ... \geq l_b$ and $u_b \geq u_{N_s}$, we get

$$\leq \gamma + \frac{u_b \cdot ((\gamma - 1) \cdot SI[b] - 1) + u_b}{b \cdot l_b}.$$

Since $SI[b] = b$ and $u_i/l_i \leq \gamma$

$$\leq \gamma + \frac{\gamma \cdot l_b \cdot (\gamma - 1) \cdot b}{b \cdot l_b} = \gamma + \gamma \cdot (\gamma - 1) = \gamma^2.$$

□

To finish Theorem 1 proof, from Equations (5) and (6) we get:

$$ub/lb \leq \frac{(\sum_{i=1}^{b-1} u_i) + u_b \cdot (SI[b+1] - SI[b]) + u_{N_s}}{\sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s-1} l_i} + \frac{\sum_{i=b+1}^{N_s-1} u_i \cdot (SI[i+1] - SI[i])}{\sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s-1} l_i}$$

From Lemma 2 and Lemma 3 we get:

$$ub/lb \leq \frac{\gamma^2(\sum_{i=1}^{b} l_i) + \gamma^2(\sum_{i=b+1}^{N_s-1} l_i)}{\sum_{i=1}^{b} l_i + \sum_{i=b+1}^{N_s-1} l_i} \leq \gamma^2$$

□

III. The general case $d > 2$

Given $d > 2$ sampled probability distributions

$$(SI, SP_1), ..., (SI, SP_d),$$

and the correct key $k^* = (k_1, ..., k_d)$, we now follow the intuition of the $d = 2$ case to solve the general case. To do so, we merge each two sampled distributions into one joint distribution, sub-sample the joint distributions, and continue in the same way until we get to a single pair of sampled distributions. We achieve this via a sequence of algorithms described below.

A. Merging two sampled distributions into a joint distribution

Given two sampled non-increasing probability distributions $(SI, SP_1), (SI, SP_2)$, each of size N_s, we wish to merge them into one non-increasing distribution, and compute lower and upper bounds on the ranks of the points. Algorithm 4 implements this task.

First, the algorithm goes over the grid of N_s^2 points (i, j) such that $1 \leq i \leq N_s$ and $1 \leq j \leq N_s$. For each point (i, j) it calculates the point's probability $SP_1[i] \cdot SP_2[j]$. Then, it sorts these points in decreasing order of their probabilities.

Given two consecutive points (i_1, j_1) and (i_2, j_2) in the sorted order such that $Prob(i_1, j_1) \geq Prob(i_2, j_2)$, all the points whose probability is greater than $Prob(i_1, j_1)$ have probability greater than $Prob(i_2, j_2)$, therefore, all the points in the rank of (i_1, j_1) are contained in the rank of (i_2, j_2). Relying on this observation, if we know the order of the N_s^2 points according to their probabilities, we can bound the accumulative rank of these points while going over them from the most likely point to the least. In this way, the upper-bound of the rank of the current point (i_c, j_c) is the upper bound of the previous point (i_p, j_p) plus the following expressions:

$$\begin{aligned} &(SI[j_p + 1] - SI[j_p]) \cdot (SI[i_p + 1] - SI[i_p] - 1) \\ &+ SI[j_p + 1] - SI[j_p] - 1 \\ &+ 1 \\ &= (SI[j_p + 1] - SI[j_p]) \cdot (SI[i_p + 1] - SI[i_p]). \end{aligned} \tag{10}$$

The first term in (10), $(SI[j_p + 1] - SI[j_p]) \cdot (SI[i_p + 1] - SI[i_p] - 1)$, represents the number of points that might come after the previous point and before the current point, which are not on the SI grid. I.e., these are the points (i, j) s.t.

$$SI[i_p] < i < SI[i_p + 1] \text{ and } SI[j_p] \leq j < SI[j_p + 1].$$

$SI[j_p + 1]$ is not included since we haven't reached that point yet.

The second term in (10), $SI[j_p+1]-SI[j_p]-1$, represents the number of points that might come after the previous point and before the current point which are on the SI grid. I.e., these are the points (i, j) s.t.

$$i = SI[i_p] \text{ and } SI[j_p] < j < SI[j_p + 1].$$

$SI[j_p]$ is not included since the point $(SI[i_p], SI[j_p])$ is the previous point and it was already included and $SI[j_p + 1]$ is not included since we haven't reached that point yet.

The last addition in (10) is 1, accounting for the current point itself.

The resulting expression can be seen in Algorithm 4 (line 13). A similar derivation can be done for the lower bound (omitted).

Algorithm 4 Calculating the joint probability distribution.

Require: Sampled probability distributions SP_1, SP_2 each of size N_s.
Ensure: Joint probability distribution.

```
1:  r = 1
2:  for i = 1 to N_s do
3:      for j = 1 to N_s do
4:          Y(r, 1) = SP_1[i] · SP_2[j];
5:          Y(r, 2) = (i, j);
6:          r = r + 1
7:      end for
8:  end for
9:  Y = Sort(Y) in decreasing order of Y(r, 1)
10: ub(1) = 1;  lb(1) = 1;  p(1) = SP_1[1] · SP_2[1]
11: for r = 2 to N_s^2 do
12:     (i_c, j_c) = Y(r, 2);  (i_p, j_p) = Y(r − 1, 2)
13:     ub(r) = ub(r − 1) + (SI(j_p + 1) − SI(j_p)) · (SI(i_p + 1) − SI(i_p))
14:     lb(r) = lb(r − 1) + (SI(j_c) − SI(j_c − 1)) · (SI(i_c) − SI(i_c − 1))
15:     p(r) = Y(r, 1)
16: end for
17: return (ub, lb, p)
```

B. Sampling the joint probability distribution

The output of Algorithm 4 is a distribution over N_s^2 elements. We now show that we can sub-sample this distribution, via exponential sampling, using the same parameters b and γ used to create the one-dimension samples. Theorem 2 below shows that a sub-sampling with the same b and γ always exists.

We would like to sample this joint probability distribution using Algorithm 2, using b and γ, except now instead of the 1-dimensional ranks we sample using the rank-upper/lower-bounds, See Algorithm 5.

For this, we shall prove in Lemma 5 that the first b indices of the joint probability distribution are $1, ..., b$ and we shall prove in Theorem 2 that the ratio between any two successive upper ranks is at most γ.

Algorithm 5 Sub-Sampling the joint distribution.

Require: A joint probability distribution (ub, lb, p) of size N_s^2, b, γ
Ensure: Sampled probability distribution (SU, SL, SP)
1: for $i = 1$ to b do
2: $SU[i] = i;\ SL[i] = i;\ SP[i] = p[i]$
3: end for
4: $j = b;\ i = j + 1;\ c = j + 1$
5: while $i < N_s^2$ do
6: if $ub[i]/ub[j] \leq \gamma$ and $ub[i+1]/ub[j] > \gamma$ then
7: $SU[c] = ub[i];\ SL[c] = ul[i];\ SP[c] = p[i]$
8: $c = c + 1;\ j = i$
9: end if
10: $i = i + 1$
11: end while
12: $SU[c] = ub[N_s^2];\ SL[c] = lb[N_s^2];\ SP[c] = p[N_s^2]$
13: return (SU, SL, SP)

Lemma 4: For any index $i \geq b + 1$ in SI it holds that

$$SI[i] - SI[i-1] \leq (\gamma - 1) \cdot SI[i-1].$$

Lemma 5: Given two sampled probability distributions (SI, SP_1) and (SI, SP_2) that are sampled by Algorithm 2 merged by Algorithm 4. The first b upper ranks in the upper joint probability distribution are the integers $1, .., b$ and the first b lower ranks in the lower joint probability distribution are the integers $1, .., b$.

Proof: According to the sampling process in Algorithm 2 it holds: $\forall i \leq b\ SI[i] = i$. Therefore, the joint probability contains the indices of $(i, j) \in \{1, ..., b\} \times \{1, ..., b\}$. Since the first b points with the highest probabilities are somewhere in the square: $\{1, ..., b\} \times \{1, ..., b\}$. The rank of the first b composed only from points in this square, therefore for $i \leq b$, the upper bound and lower bound of the i'th element in the joint distribution are equal to each other and equal to i.

Theorem 2: Given the joint probability distribution of the sampled probability distributions $(SI, SP_1), (SI, SP_2)$, The ratio between any two consecutive upper bound ranks is at most γ, where $1 < \gamma \leq 2$.

Proof: Let $up(i_c, j_c)$ be the upper bound on the rank of point (i_c, j_c) as in Algorithm 4. As can be seen in Algorithm 4 the difference between the upper bound ranks of any two consecutive points (i_c, j_c) and (i_p, j_p) is

$$up(i_c, j_c) - up(i_p, j_p) = (SI(j_p + 1) - SI(j_p)) \cdot (SI(i_p + 1) - SI(i_p)).$$

By Lemma 4 it holds that

$$SI(j_p + 1) - SI(j_p) \leq SI(j_p) \cdot (\gamma - 1)$$
$$SI(i_p + 1) - SI(i_p) \leq SI(i_p) \cdot (\gamma - 1).$$

Therefore,

$$up(i_c, j_c) - up(i_p, j_p) \leq SI(j_p) \cdot SI(i_p) \cdot (\gamma - 1)^2$$

The trivial lower bound of $rank(i_p, j_p)$ is the multiplication of its indexes $SI(i_p)$ and $SI(j_p)$, therefore

$$up(i_n, j_n) - up(i_p, j_p) \leq up(i_p, j_p) \cdot (\gamma - 1)^2$$

Since $1 < \gamma \leq 2$, it holds $(\gamma - 1)^2 \leq (\gamma - 1)$, therefore

$$up(i_c, j_c) - up(i_p, j_p) \leq up(i_p, j_p) \cdot (\gamma - 1)$$

and we get

$$up(i_c, j_c) \leq up(i_p, j_p) \cdot \gamma.$$

□

Theorem 2 shows that in the joint $N_s \times N_s$ distribution, the upper bounds of every two consecutive points (in sorted order) obey the invariant $ub(i_c, j_c) \leq \gamma \cdot ub(i_p, j_p)$.

Corollary 1: The sample produced by Algorithm 5 on an input distribution of size N_s^2 consists of $O(N_s)$ ranks.

C. The ESrank Algorithm: Putting it all together

Given $d > 2$ sampled probability distributions

$$(SI, SP_1), ..., (SI, SP_d)$$

, and the correct key $k^* = (k_1, ..., k_d)$, we first merge the d sampled probability distributions into 2 sampled joint distributions in the following way: we merge the first two sampled distributions $[SI, SP_1]$ and $[SI, SP_2]$ into one sampled $[SU_{12}, SL_{12}, SP_{12}]$. Next we merge the results with $[SI, SP_3]$ using Algorithm 6 which is the extended version of Algorithm 4. We continue in the same way until the first $d - 1$ sampled distributions are merged into one sampled joint distribution. Now, we apply Algorithm 7 which is the extended version of Algorithm 3 on the sampled joint distribution of the first $d-1$ sampled distributions $[SU_{1...d}, SL_{1...d}, SP_{1...d}]$ and $[SI, SP_d]$ to get the upper and the lower bound of $Rank(k^*)$. Algorithm 8 shows the complete pseudo-code for ESrank.

D. Theoretical Performance

1) Time complexity: At each level of Algorithm 8 it uses Algorithm 4 to merge the sampled distributions received from the previous level. Algorithm 4 goes over N_s^2 pairs, calculates their probabilities using $\Theta(N_s^2)$ time, and sorts them using $\Theta(N_s^2 \cdot \log N_s)$ time. Let $T(d, \gamma)$ be the total the running time. Then

$$\begin{aligned} T(d, \gamma) &\leq \sum_{i=1}^{d-2} i(\log_\gamma N)^2 \log\left(i(\log_\gamma N)^2\right) \\ &\leq d^2(\log_\gamma N)^2 \log(d \log_\gamma N) \\ &= O(d^{2+\epsilon}(\log_\gamma N)^{2+\epsilon}) \text{ for } 0 < \epsilon < 1. \end{aligned}$$

I.e., we see that ESrank has a poly-logarithmic time complexity (in N).

Algorithm 6 Calculating the extended joint probability distribution

Input: Sampled probability distributions $[SI, SP_1]$, $[SU_2, SL_2, SP_2]$ each of size N_s.
Output: Joint probability distribution.

$r = 1$
for $i = 1$ to N_s do
  for $j = 1$ to N_s do
    $Y(r,1) = SP_1[i] \cdot SP_2[j]$; $Y(r,2) = (i,j)$
    $r = r + 1$
  end for
end for
$Y = Sort(Y)$ in decreasing order of $Y(r,1)$
$ub(1) = 1$; $lb(1) = 1$; $p(1) = SP_1[1] \cdot SP_2[1]$
for $r = 2$ to N_s^2 do
  $(i_c, j_c) = Y(r,2)$; $(i_p, j_p) = Y(r-1,2)$
  $ub(r) = ub(r-1) + (SU_2(j_p+1) - SU_2(j_p)) \cdot (SI(i_p+1) - SI(i_p)$
  $lb(r) = lb(r-1) + (SL_2(j_c) - SL_2(j_c-1)) \cdot (SI(i_c) - SI(i_c-1)$
  $p(r) = Y(r,1)$
end for
return (ub, lb, p)

2) Accuracy: Assume the correct key is $k^* = (k_1, ..., k_d)$. For a key (k_i, k_{i+1}) and $(SI, SP_i), (SI, SP_{i+1})$ let $k_{i,i+1}$ be the real rank of (k_i, k_{i+1}). At the lowest level Theorem 1 and Algorithm 3 give that $up(k_i, k_{i+1}) \leq \gamma^2 k_{i,i+1}$. In the next level, each rank in the sampled joint distribution is multiplied by at most γ^2, therefore each term in the sum that composes $up(\gamma^2 k_{i,i+1}, k_{i+2})$ is multiplied by at most γ^2. Hence $up(\gamma^2 k_{i,i+1}, k_{i+2}) \leq \gamma^2 up(k_{i,i+1}, k_{i+2}) \leq \gamma^2\gamma^2 k_{i,i+1,i+2} = \gamma^4 k_{i,i+1,i+2}$. We continue in the same way, and get

$$up(Rank(k^*))/Rank(k^*) \leq \prod_{i=1}^{d-1} \gamma^2 = \gamma^{2d-2}. \quad (11)$$

Since $rank(k^*)$ might be any value in

$$[low(Rank(k^*), up(Rank(k^*),]$$

we get

$$accuracy(d, \gamma) = up(Rank(k^*))/low(Rank(k^*)) \leq \gamma^{2d-2}.$$

E.g., for AES-128 with a preprocessing step of merging the 16 8-bit distributions into $d = 8$ 16-bit distributions we get $2d - 2 = 14$.

3) Space complexity: In the first step we need to store d distributions of size $\log_\gamma N$ from Algorithm 2. In order to merge each pair of distributions into one, we need additional memory of $(\log_\gamma N)^2$. After merging 2 distributions each of size $(\log_\gamma N)$, we get one sampled distribution of size $(\log_\gamma N^2)$ which is $2(\log_\gamma N)$. Since we don't need the original pair anymore, we can overwrite this space of size $2(\log_\gamma N)$ and store the new distribution into it. Therefore, the sampled joint distribution of size $2(\log_\gamma N)$ overwrites the two distributions each of size $(\log_\gamma N)$. In the next step, we merge two distributions, one of size $2\log_\gamma N$ and one of size $\log_\gamma N$, therefore we need additional space of $2(\log_\gamma N)^2$. Again, the new sampled joint distribution of size $3\log_\gamma N$ overwrites the two distributions of size $\log_\gamma N$ and $2\log_\gamma N$. In the last step, we need to merge 2 distributions, one of size $(d-2)\log_\gamma N$ and one of size $\log_\gamma N$, therefore the maximum additional space we need is $(d-2)(\log_\gamma N)^2$. In total we get

$$space(d, \gamma) = d\log_\gamma N + (d-2)(\log_\gamma N)^2 = O(d(\log_\gamma N)^2).$$

Algorithm 7 Calculating Upper and Lower bounds - Extended version.

Input: Sampled probability distributions $[SI, SP_1]$, $[SU_2, SL_2, SP_2]$ each of size N_s, b, the correct key $k^* = (k_1, k_2)$ probability p^*.
Output: Upper and lower bounds on $Rank(k^*)$.

$i = 1$; $j = N_s$; $ub = 0$; $lb = 0$
while $i \leq N_s$ and $j \geq 1$ do
  $pCurr = SP_1[i] \cdot SP_2[j]$
  if $pCurr \geq p^*$ then
    $u = SU_2[j]$; $l = SL_2[j]$
    $ub = ub + u$; $lb = lb + l$
    if $i \geq b + 1$ then
      $ub = ub + uPrev \cdot (SI[i] - SI[i-1] - 1)$
      $lb = lb + l \cdot (SI[i] - SI[i-1] - 1)$
    end if
    $i = i + 1$; $uPrev = u$
  else if $j > 1$ then
    $pNext = SP_1[i] \cdot SP_2[j-1]$
    if $pNext < p^* < pCurr$ then
      $u = SU_2[j] - 1$; $l = SL_2[j-1]$
      $ub = ub + u$; $lb = lb + l$
      if $i \geq b + 1$ then
        $ub = ub + uPrev \cdot (SI[i] - SI[i-1] - 1)$
        $lb = lb + l \cdot (SI[i] - SI[i-1] - 1)$
      end if
      $i = i + 1$; $uPrev = u$
    else
      $j = j - 1$
    end if
  else
    $j = j - 1$
  end if
end while
if $j < 1$ and $i \leq N_s$ then
  $ub = ub + uPrev \cdot (SI[i] - SI[i-1] - 1)$
end if
return (lb, ub)

E. How to select the parameter Gamma

One of the significant advantageous of our ESrank algorithm is that the error is bounded and it can be tuned

Algorithm 8 ESrank: Calculating the upper and lower bounds for $d > 2$.

Input: The probability distributions $P_1, ..., P_d$, the correct key $k^* = (k_1, ..., k_d)$ probability p^*, b and γ.
Output: Upper and lower bounds of $rank(k^*)$.

```
for i = 1 to d do
  (SI, SP_i) = Alg2(P_i, b, γ) {Sample the input distributions}
end for
(SU_c, SL_c, SP_c) = (SI, SI, SP_1)
for i = 2 to d − 1 do
  (ub_c, lb_c, p_c) = Alg6((SI, SP_i), (SU_c, SL_c, SP_c)) {Merge}
  (SU_c, SL_c, SP_c) = Alg5(ub_c, lb_c, p_c, b, γ) {SubSample}
end for
(ub, lb) = Alg7((SI, SP_d), (SU_c, SL_c, SP_c), b, p*) {Calculate bounds on rank}
return (ub, lb)
```

to the desired accuracy. Therefore, if we want accuracy of λ, i.e., that the ratio between the the upper bound and the lower bound in number of bits will be λ, then we should solve the following equation:

$$\gamma^{2d-2} = \lambda.$$

This leads us to the desired γ:

$$\gamma = \lambda^{1/(2d-2)}. \tag{12}$$

For example, to get 1-bit accuracy we need to use $\lambda = 2$, and if $d = 5$ then γ is 1.09.

IV. The Unbounded Error of the Histogram Algorithm

The currently best rank estimation algorithm is the Histogram algorithm of [14], [25]. The authors did not provide a bound on its estimation error: in this section we demonstrate that its error is in fact unbounded. We do so by constructing an infinite family of probability distributions for which the ratio between the Histogram upper- and lower-bounds on a key's rank can be arbitrarily large: effectively the gap between the lower and upper bounds will be a multiplicative factor of N or more.

To explain our construction we need to briefly review the Histogram algorithm (see Algorithm 9). The algorithm receives as input d probability distributions $\{P_i\}_{i=1}^d$. Let

$$l_{min} = \min_{i=1}^{d} \min_{j=1}^{N} \log P_i[j]$$

and define the log-probabilities as

$$LP_i[j] = \log P_i[j] - l_{min}.$$

Notice that the log-probabilities obey $LP_i[j] \geq 0$ for all i, j. Let

$$l_{max} = \max_{i=1}^{d} \max_{j=1}^{N} LP_i[j].$$

For a given number of bins B and for each LP_i, define its corresponding histogram with B equally-sized bins as

$$H_i = hist(LP_i, B, l_{max}).$$

Let the bin width be

$$w = l_{max}/B.$$

The Histogram algorithm sequentially executes a convolution between all the histograms to obtain a final histogram of dB bins denoted by H_F. For a given secret key $k^* = (k_1, ..., k_d)$ it calculates the bin of H_F corresponding to k^* by

$$b^* = \frac{1}{w} \sum_{i=1}^{d} LP_i[k_i].$$

Finally, based on $H_F[b^*]$, it returns upper and lower bounds of the given key's rank by summing the weight of the bins in H_F either with, or without, the ball of radius $d/2$ around b^*.

Algorithm 9 Histogram Algorithm.

Input: The d vectors of log probabilities $\{LP_i\}_{i=1}^d$, the number of bins B, and the secret key indices $k^* = (k_1, ..., k_d)$
Output: upper and lower bounds on the rank of k^*

```
1: l_max = max(max(LP_0), ..., max(LP_d))
2: w = l_max / B
3: h_0 = hist(LP_0, B, l_max)
4: h_1 = hist(LP_1, B, l_max)
5: H_F = conv(h_0, h_1) {convolution}
6: for i = 2 to d do
7:    h_i = hist(LP_i, B, l_max)
8:    H_F = conv(h_i, H_F)
9: end for
10: b* = ⌊(1/w) Σ_{i=1}^{d} (LP_i(k_i))⌋
11: bin_lower = min(b* + ⌈d/2⌉ + 1, size(H_F))
12: bin_upper = max(b* − ⌈d/2⌉, 0)
13: upper = Σ_{i=bin_upper}^{size(H_F)} H_F[i]
14: lower = Σ_{i=bin_lower}^{size(H_F)} H_F[i]
15: return [upper, lower]
```

As can be seen in Algorithm 9, the accuracy of the Histogram rank estimation depends on $H_F[i]$ for each $i \in [b^* - d/2, b^* + d/2]$, which means that it depends on the probability distribution.

Our construction is such that almost all the the probabilities are mapped to the same bin. Intuitively, in the first dimension, we set the highest probability, $P_1[1]$, to be some value x, and divide the remaining $1 - x$ probability mass almost equally among the other $N-1$, so each of the smaller probabilities has a value of $y \approx (1-x)/(N-1)$ (within a range of size ϵ). If the range of the probabilities $y \approx (1-x)/(N-1)$, after converting them to log probabilities, is smaller than w, then they will be mapped to the same bin, therefore the histogram of this dimension

will have exactly two non-zero bins: one bin with $N-1$ elements, and one bin with one element. For all the other dimensions $j \in [2,d]$ we set the probabilities to be $P_j[r] \approx 1/N$, so all the probability mass falls into a single bin in its dimension's histogram. Below we work out the technical constraints on the choice of the values of x and ϵ as functions of N and B.

For a given B and N, choose x such that:

$$x \geq \frac{(1-\frac{1}{B})\cdot(\frac{B+1}{B-1})^B}{N-1+(1-\frac{1}{B})\cdot(\frac{B+1}{B-1})^B} \xrightarrow{B\to\infty} \frac{e^2}{N-1+e^2}$$

e.g., if $N \geq 10$ we can use $x = \frac{1}{2}$. Let y be:

$$y = \frac{1-x}{N-1}$$

and let ϵ be

$$\epsilon = \frac{2(1-x)}{B(N-1)(N-2)}.$$

This value of ϵ guarantees that the range of the probabilities $y \approx (1-x)/(N-1)$, after converting them to log probabilities, is in fact smaller than w, therefore they will be mapped to the same bin. Define P as the following probability distribution of the first dimension:

$$P = [x, y+\epsilon(\frac{N-2}{2}), ..., y, ..., y-\epsilon(\frac{N-2}{2})]$$

i.e., all the probabilities except for the largest are almost equal to y, and they span a range of $\epsilon(N-2)$. Clearly P sums to 1.

We can see that the range of the smallest positively shifted log probabilities, i.e., $LP_1[2]$ - $LP_1[N]$ is bounded by the bin width, i.e., $w = LP_1[1]/B$:

$$\log\left[\frac{y+\epsilon\left(\frac{N-2}{2}\right)}{y-\epsilon\left(\frac{N-2}{2}\right)}\right] \leq w = \log\left[\left(\frac{x}{y-\epsilon\left(\frac{N-2}{2}\right)}\right)^{\frac{1}{B}}\right]$$

Therefore, as planned, the $(N-1)$ smallest probabilities of the first dimension are mapped to the same first bin, and we get the following histogram:

$$H_1[0] = N-1, H_1[B-1] = 1, H_1[i] = 0 \text{ for } i \in [1, B-2]$$

For dimension $j \in [2,d]$ the histogram is

$$H_j[a] = N, H_j[i] = 0 \text{ for } i \neq a$$

where a is the corresponding bin of the probability $1/N$. After all $d-1$ convolutions we get the following H_F:

$$H_F[(d-1)\cdot a] = (N-1)N^{d-1}, H_F[(d-1)\cdot a+(B-1)] = N^{d-1}$$

$$H_F[i] = 0 \text{ for all other indexes } i.$$

For the $(N-1)N^{d-1}$ keys $k^* = (k_1, ..., k_d)$ s.t $k_1 \neq 1$, the Histogram algorithm error ratio is

$$\frac{upper}{lower} = \frac{N^d}{N^{d-1}} = N$$

and for the N^{d-1} keys $k^* = (k_1, ..., k_d)$ s.t. $k_1 = 1$, the Histogram algorithm error ratio is not defined since the lower bound is 0 and the upper bound is N^{d-1}. I.e., the error is unbounded/undefined for any user-selected parameter B.

We conclude that since Histogram depends on the probability distribution, there are cases in which its accuracy is poor.

In contrast, ESrank has a bounded error which does not depend on the probability distribution or on N. As shown by equation (11), the error ratio is always bounded by γ^{2d-2} for any d probability distributions, and γ can be chosen to achieve the desired accuracy, for any probability distribution.

V. Empirical Evaluation

We evaluated the performance of the ESrank algorithm through an extensive simulation study. We compared our algorithm to the currently best rank estimation algorithm: the Histogram algorithm of [14]. We implemented both in Matlab. We published our ESrank Matlab code in GitHub [3]. We ran both algorithms on a 2.80GHz i7 PC with 8GB RAM running Microsoft Windows 7, 64bit. The later experiments, with the password strength corpus, were run on the same computer after its RAM was increased to 32GB.

For the performance evaluation we used two different types of data: data from a side-channel attack [12] and data from a password strength estimator [9].

A. Side channel attack corpus

Within this data corpus there are 611 probability distribution sets gathered from a specific SCA. The SCA of [12] was against AES [1] with 128-bits keys running on an embedded processor with an unstable clock. Each set represents a particular 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. We used the same technique suggested in [14]: merge the $d = 16$ probability lists of size $N = 2^8$ into $d = 8$ lists of size $N = 2^{16}$. We measured the upper bound, lower bound, time and space for each trace using ESrank and the Histogram rank estimation.

1) Bound Tightness: Figure 2 shows that the analytical performance of equation (11) indeed agrees with the empirical results. For different values of γ we get accuracy which corresponds to at most γ^{14}: e.g., when $\gamma = 1.055$ Figure 2 shows a margin of at most 1 bit. We can see that as γ becomes closer to 1, the accuracy becomes closer to 0. As we expected, the maximum gap between the upper bound and the lower bound happens for ranks around $100-120$ since the difference between any two successive indices in the sampled set becomes greater when the indices become greater.

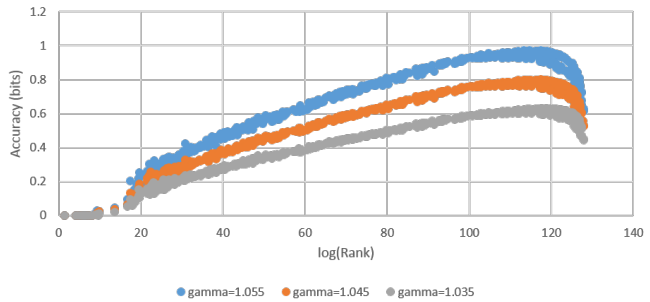

Fig. 2. The accuracy ($\log_2$ of the ratio between the upper- and lower-bounds) for the ESrank algorithm as a function of $\log_2(Rank(k^*))$ for different parameter settings: $\gamma = 1.055$ (blue), $\gamma = 1.045$ (red), $\gamma = 1.035$ (gray).

	Time (Seconds)	Space (MB)	Accuracy < 1 bit (%)
$\gamma = 1.035$	0.603	5.01	100
$\gamma = 1.045$	0.356	3.06	100
$\gamma = 1.055$	0.231	2.07	100
$B = 55K$	0.757	3.52	100
$B = 45K$	0.524	2.88	100
$B = 35K$	0.307	2.24	100

TABLE I
Performance summary of the ESrank and Histogram algorithms in the side-channel corpus. The Accuracy column indicates the percentage of traces for which the difference between the upper- and lower-bounds of the estimated ranks was below 1 bit.

2) Time and Space Analysis: Table I shows the time, the space and the percentage of the traces for which the accuracy is better than 1 bit, for ESrank with $\gamma = 1.035, 1.045, 1.055$ and for Histogram [14] with B =55,000, 45,000, 35,000. As we can see, the two algorithms, using the described parameters - all take less than 0.8 seconds and use under 5.5 MB of memory. In a practical sense ESrank is on-par with the Histogram algorithm: both exhibit a run-time of under 1 second using less than 5.5 MB, to get a 1-bit margin of uncertainty in the rank for all ranks up to 2^{128} on this corpus.

B. Password strength estimator corpus

In this corpus there are 5 probability distributions, each representing one dimension in the password strength estimator of [9]. The password strength estimator receives as an input a password probability. Based on the five probability lists, it returns the rank of the received password (see Table II). Each one of the five probability distributions is sorted in decreasing order.

Note that this corpus is qualitatively different from the SCA corpus: the dimension lengths are very skewed, with the baseword dimension being 5 orders of magnitude larger than the l33t dimension. Furthermore, the dimensions have many "ties": e.g., in the baseword dimension there are 999,533 words with probability of exactly $5.521 \cdot 10^{-10}$. The equal-probability values make the distributions "lumpy"

	dimension	length
1	prefix	7,136,112
2	base word	221,271,911
3	suffix	10,933,957
4	capitalization	239,626
5	l33t	2,704

TABLE II
Password strength estimator of [9] dimension.

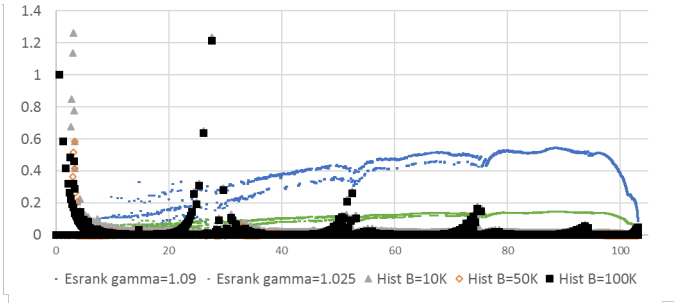

Fig. 3. The accuracy ($\log_2$ of the ratio between the upper- and lower-bounds) for the ESrank algorithm as a function of $\log_2(Rank(k^*))$ for different parameter settings: $\gamma = 1.025$ (green), $\gamma = 1.09$ (blue) and the Histogram algorithm as a function of $\log_2(Rank(k^*))$ for different parameter settings: B = 10K (grey), B = 50K (orange) and B = 100K (black). The grey and orange symbols are mostly obscured by the black $B = 100$K symbols .

with long "plateaux" sections, and as we shall see, such distributions stress rank estimation algorithms.

Since only the last step of ESrank depends on the input password probability and since the password strength estimator always uses ESrank with the same five probability distributions, we can divide the algorithm into two steps: a preprocessing step and a lookup step. The preprocessing is executed only once, and the lookup step is executed for any input password probability. The preprocessing step includes the merge steps till we get the two last merged sampled probability distributions. The lookup step includes the calculation of the upper and the lower bounds of the received password probability based on the two probability lists.

For comparison, we adapted the Histogram algorithm to work with dimensions of different lengths. We also divided the Histogram rank estimation into a preprocessing step and a lookup step. The preprocessing step includes the convolutions to get the final histogram and also includes the calculation of an accumulated sum histogram. Then, in lookup step, given a password, the correct bin index b^* is calculated and the corresponding lower and upper bounds are returned.

We went over the probabilities of the possible passwords in the volume and for each password we calculated the preprocessing time, the lookup time, the preprocessing space, the lookup space, the accuracy (number of bits of $upper/lower$) using the ESrank and the Histogram algorithms.

Figure 3 shows the accuracy of both algorithms with

	preprocessing Time (Seconds)	Lookup Time (μSeconds)	preprocessing Space (MB)	Lookup Space (MB)	Accuracy (bits)
$\gamma = 1.025$	4.4473	25.032	7.088	0.024	0.147
$\gamma = 1.09$	4.1621	9.515	0.678	0.007	0.547
$\gamma = 1.15$	4.1271	7.104	0.275	0.044	0.870
$B = 100K$	3.0691	0.273	4	4	1.215
$B = 50K$	3.1494	0.346	2	2	1.216
$B = 10K$	2.8072	0.399	0.4	0.4	1.263

TABLE III
Performance summary of the ESrank and Histogram algorithms in the password corpus.

different setting. For this corpus with $d = 5$, to obtain an accuracy of 1-bit, equation (11) requires

$$\gamma^{2d-2} = \gamma^8 \leq 2$$

so $\gamma \leq 1.09$. As can be seen in the figure, $\gamma = 1.09$ gives accuracy noticeable better than 1 bit. The curves in Figures 3 show the same pattern we observed on the SCA corpus in Figure 2: the best accuracy is achieved for the lowest- and highest-ranks and the poorest accuracy, closest to the bound γ^{2d-2}, occurs near the high ranks. We also see that decreasing γ to $\gamma = 1.025$ improves the accuracy for all ranks. The double "trail" of accuracy levels for ranks up to 75 bits is due to the fact that the distributions are "lumpy" with many ties. All the k^*'s whose value k_i has the same probability behave equally for this dimension - so we can observe a "trail" for passwords for which this occurs and another trail when it does not.

Figure 3 also shows the accuracy of the Histogram algorithm, for 3 choices of B. First, the figure shows that the worst-case error is relatively high (1.2 bits). But more importantly, the figure also shows that poor accuracy which does not vanish can be observed for the lowest ranks, and also for ranks around $2^{30}, 2^{50}, 2^{75}, 2^{90}$, again due to the many ties in the distributions. While the empirical accuracy is not as catastrophic as in the extreme distribution constructed in Section IV, Figure 3 does demonstrate that increasing B does not guarantee an improved accuracy. This is in contrast with ESrank that enjoys a bounded error and whose accuracy improves with smaller gamma.

In Table III we can see the preprocessing time, the preprocessing space, the lookup time, the lookup space and the accuracy of the ESrank and the histogram in different parameters. As can be seen, the ESrank is on-par with the Histogram algorithm in terms of time and space, however, we can see that the maximum error in the Histogram is stable at 1.2 bits and is not significantly changed with larger number of bins B.

VI. Conclusion

In this paper we proposed a simple and effective new rank estimation method called ESrank. It is the first rank estimation algorithm that enjoys provable poly-logarithmic time- and space-complexity, which has bounded error ratio of γ^{2d-2}, for any probability distributions, where d is the number of dimensions and γ can be chosen according to the desired accuracy.

Our main idea is to use exponential sampling to drastically reduce the algorithm's complexity. We have rigorously analyzed its accuracy, and its time and space complexities. We proved ESrank has a poly-logarithmic time- and space-complexity, and it can be driven to any desired level of accuracy (trading off time and space against accuracy).

We then compared ESrank to the currently-best histogram based algorithm. We show that the Histogram's accuracy depends on the probability distribution, and that these exist distributions for which its estimation error is unbounded.

After rigorously bounding the accuracy, time and space complexities, we evaluated the performance of ESrank on a real SCA data corpus and on a password strength corpus, and compared it to the histogram-based algorithm. We show that ESrank gives excellent rank estimation (with roughly a 1-bit margin between lower and upper bounds), with a performance that is on-par with the Histogram algorithm. On the password strength corpus we find that Histogram's estimation error cannot be made to vanish for any choice of number of Histogram bins. Hence ESrank is an important and 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] Agrawal, D., Archambeault, B., Rao, J., and Rohatgi, P. The EM side-channel(s). In Cryptographic Hardware and Embedded Systems-CHES 2002. 2003, pp. 29–45.

[3] Anonymous. ESrank Matlab implementation, 2019.

[4] Bentley, J. L., and Yao, A. C.-C. An almost optimal algorithm for unbounded searching. Information processing letters 5, 3 (1976), 82–87.

[5] Bernstein, D. J., Lange, T., and van Vredendaal, C. Tighter, faster, simpler side-channel security evaluations beyond computing power. IACR Cryptology ePrint Archive 2015 (2015), 221.

[6] Bogdanov, A., Kizhvatov, I., Manzoor, K., Tischhauser, E., and Witteman, M. Fast and memory-efficient key recovery in side-channel attacks. In Selected Areas in Cryptography (SAC) (2015).

[7] Choudary, M. O., and Popescu, P. Back to massey: Impressively fast, scalable and tight security evaluation tools. In International Conference on Cryptographic Hardware and Embedded Systems (2017), Springer, pp. 367–386.

[8] David, L., and Wool, A. 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 (San Francisco, Feb. 2017), Springer Verlag, pp. 311–327.

[9] David, L., and Wool, A. Online password guessability via multi-dimensional rank estimation. https://arxiv.org/abs/1912.02551.

[10] Dell'Amico, M., and Filippone, M. Monte carlo strength evaluation: Fast and reliable password checking. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015), ACM, pp. 158–169.

[11] Duc, A., Faust, S., and Standaert, F.-X. Making masking security proofs concrete. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (2015), Springer, pp. 401–429.

[12] Fledel, D., and Wool, A. Sliding-window correlation attacks against encryption devices with an unstable clock. In Proc. 25th Conference on Selected Areas in Cryptography (SAC) (Calgary, Aug. 2018).

[13] Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic analysis: Concrete results. In Cryptographic Hardware and Embedded Systems—CHES 2001 (2001), Springer, pp. 251–261.

[14] Glowacz, C., Grosso, V., Poussier, R., Schueth, J., and Standaert, F.-X. Simpler and more efficient rank estimation for side-channel security assessment. In Fast Software Encryption (2015), pp. 117–129.

[15] Grosso, V. Scalable key rank estimation (and key enumeration) algorithm for large keys. In International Conference on Smart Card Research and Advanced Applications (2018), Springer, pp. 80–94.

[16] Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In Advances in Cryptology—CRYPTO'99 (1999), Springer, pp. 388–397.

[17] Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Advances in Cryptology—CRYPTO'96 (1996), pp. 104–113.

[18] Li, Y., Meng, X., Wang, S., and Wang, J. Weighted key enumeration for em-based side-channel attacks. In 2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC) (2018), IEEE, pp. 749–752.

[19] Li, Y., Wang, S., Wang, Z., and Wang, J. A strict key enumeration algorithm for dependent score lists of side-channel attacks. In International Conference on Smart Card Research and Advanced Applications (2017), Springer, pp. 51–69.

[20] Longo, J., Martin, D. P., Mather, L., Oswald, E., Sach, B., and Stam, M. How low can you go? using side-channel data to enhance brute-force key recovery. IACR Cryptology ePrint Archive 2016 (2016), 609.

[21] Martin, D. P., Mather, L., and Oswald, E. Two sides of the same coin: counting and enumerating keys post side-channel attacks revisited. In Cryptographers' Track at the RSA Conference (2018), Springer, pp. 394–412.

[22] Martin, D. P., Mather, L., Oswald, E., and Stam, M. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In International Conference on the Theory and Application of Cryptology and Information Security (2016), Springer, pp. 548–572.

[23] Martin, D. P., O'Connell, J. F., Oswald, E., and Stam, M. Counting keys in parallel after a side channel attack. In Advances in Cryptology–ASIACRYPT 2015. Springer, 2015, pp. 313–337.

[24] Pan, J., Van Woudenberg, J. G., Den Hartog, J. I., and Witteman, M. F. Improving dpa by peak distribution analysis. In International Workshop on Selected Areas in Cryptography (2010), Springer, pp. 241–261.

[25] Poussier, R. Key Enumeration, Rank Estimation and Horizontal Side-Channel Attacks. PhD thesis, ICTEAM Institute, 2017.

[26] Poussier, R., Grosso, V., and Standaert, F.-X. Comparing approaches to rank estimation for side-channel security evaluations. In International Conference on Smart Card Research and Advanced Applications (CARDIS) (2015), Springer, pp. 125–142.

[27] Poussier, R., Standaert, F.-X., and Grosso, V. Simple key enumeration (and rank estimation) using histograms: an integrated approach. In Proc. 18th Cryptographic Hardware and Embedded Systems–CHES 2016. Springer, 2016, pp. 61–81.

[28] Quisquater, J.-J., and Samyde, D. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Smart Card Programming and Security. Springer, 2001, pp. 200–210.

[29] Shepherd, D. Quantum key search with side channel advice. In Selected Areas in Cryptography–SAC 2017: 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers (2018), vol. 10719, Springer, p. 407.

[30] Veyrat-Charvillon, N., Gérard, B., Renauld, M., and Standaert, F.-X. An optimal key enumeration algorithm and its application to side-channel attacks. In International Conference on Selected Areas in Cryptography (2012), Springer, pp. 390–406.

[31] Veyrat-Charvillon, N., Gérard, B., and Standaert, F.-X. Security evaluations beyond computing power. In Advances in Cryptology–EUROCRYPT 2013. Springer, 2013, pp. 126–141.

[32] Wang, S., Li, Y., and Wang, J. 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), IEEE, pp. 19–24.

[33] Ye, X., Eisenbarth, T., and Martin, W. Bounded, yet sufficient? how to determine whether limited side channel information enables key recovery. In Smart Card Research and Advanced Applications (CARDIS). 2014, pp. 215–232.