A Bounded-Space Near-Optimal Key Enumeration Algorithm for Multi-Dimensional Side-Channel Attacks

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. Enumeration of cryptographic keys in order of likelihood based on side-channel leakages has a significant importance in cryptanalysis. Previous algorithms enumerate the keys in optimal order, however their space complexity is $\Omega(n^{d/2})$ when there are d subkeys and n candidate values per subkey. We propose a new key enumeration algorithm that has a space complexity bounded by $O(d^2w + dn)$, when w is a design parameter, which allows the enumeration of many more keys without exceeding the available space. The trade-off is that the enumeration order is only near-optimal, with a bounded ratio between optimal and near-optimal ranks.
Before presenting our algorithm we provide bounds on the guessing entropy of the full key in terms of the easy-to-compute guessing entropies of the individual subkeys. We use these results to quantify the near-optimality of our algorithm's ranking, and to bound its guessing entropy.
We evaluated our algorithm through extensive simulations. We show that our algorithm continues its near-optimal-order enumeration far beyond the rank at which the optimal algorithm fails due to insufficient memory, on realistic SCA scenarios. Our simulations utilize a new model of the true rank distribution, based on long tail Pareto distributions, that is validated by empirical data and may be of independent interest.

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 timing [12], power consumption [11], electromagnetic emulation [18], electromagnetic radiation [2, 9], and other sources, can be exploited by SCA to break cryptosystems.

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. 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.

Much attention has been paid to the "divide" part of side channel analysis, aiming to optimize its performance: Kocher et al.'s Differential Power Analysis (DPA) [11], Brier et al.'s Correlation Power Analysis (CPA) [6] and Chari et al.'s Template Attacks [7] are some examples. In contrast, less attention has been paid to the "conquer" part.

1.2 Related work

The problem of merging two lists of subkey candidates was encountered by Junod and Vaudenay [10]. The simple approach of merging and sorting the subkeys lists was tractable thanks to the small size of the lists (up to 2^{13}). By decreasing the order of the probabilities, given partial information obtained for each key bit individually, Dichtl [8] considered a faster enumeration of key candidates. A more general and challenging problem is enumerating keys from lists that cannot be merged, exploiting any partial information on subkeys. For this, a probabilistic algorithm was proposed in [14]. In this work the attacker has no access to the subkey distributions but is able to generate them. The proposed solution is to enumerate keys by randomly choosing subkeys according to these distributions. This implementation requires $O(1)$ memory but most keys may be chosen many times, leading to useless repetitions.

A deterministic enumeration algorithm was described by Pan et al [16]. It enumerates key candidates in the optimal order, but large memory requirements prevent the application of this, when the number of keys to enumerate increases.

A best optimal algorithm was proposed by Veyrat-Charvillon, Gérard, Renauld and Standaert, [20], which we denote by OKEA. This algorithm significantly improves the time and memory complexity thanks to clever data structures and a recursive decomposition of the problem. 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 the 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.

Recently two improved key enumeration algorithms were proposed by Bogdanov et al. [5] and Martin et al. [13]. Similar to us, both papers improve upon OKEA [20] by suggesting bounded-memory algorithms.

Bogdanov et al. [5] uses a score-based enumeration, rather than the probability-based enumeration that OKEA and our algorithm use, producing an enumeration that is suboptimal in terms of output order. The focus of Martin et al. [13] is on rank estimation and parallelization, via a reduction to #knapsack. Like [5] they also manipulate the side-channel leakages, but into different weights.

However, neither paper provides any analytical bound on the distance between their order and the optimal order: they only provide empirical evidence based on one dataset. Further, both use additive scoring (the scores of different subkeys are added to score a full key): [5] suggests scores that are scaled-and-truncated probabilities, whereas [13] skirts this issue. This makes it difficult to compare apples to apples: the quality of their order would have been comparable to the optimal (OKEA) order and to our order only if they had used log-probabilities (whose addition is semantically equivalent to multiplication of probabilities). Moreover, with scores, standard metrics such as the Guessing Entropy, which we analyze, cannot be computed, since they require probabilities. Finally, giving our algorithm more memory greatly improves its order quality and its runtime, whereas their algorithms do not.

Ye et al. [22] take a different approach: they limit the key enumeration to a hypercube of the top e candidates for every subkey. They do not explain how to enumerate inside the hypercube. Their KSF fails if the true key is outside this hypercube. This is unlike all previously mentioned papers, which always find the correct key if given enough time. In some sense KSF is analogous to the first step of our algorithm: instead of giving up, our algorithm continues to adjacent volumes wrapping the hypercube, and uses the OKEA inside the hypercube and in the adjacent volumes, while maintaining a bound on the memory complexity.

The paper of Poussier et al. [17] is primarily a taxonomy and comparison of rank estimation algorithms, suggesting new algorithmic combinations. It continues the work of Veyrat [21] and Bernstein [4], and also of Martin et al. [13]. Rank estimation is a closely related, yet different, question, to the key enumeration we address: It doesn't necessarily require to enumerate all the keys candidates ranked before the correct key, as it is only necessary to estimate how many there are.

1.3 Contributions

We propose a new key enumeration with bounded memory requirement, which allows the enumeration of a large number of keys without exceeding the available space. The trade-off is that the enumeration order is only near-optimal, with a bounded ratio between optimal and near-optimal ranks. Our algorithm has space complexity of $O(d^2w + dn)$ where w is a design parameter.

Another contribution is an extension to the evaluation framework [19]. Before presenting our algorithm we provide bounds on the guessing entropy of the full key in terms of the easy-to-compute guessing entropies of the individual subkeys. We use these results to quantify the near-optimality of our algorithm's ranking, and to bound its guessing entropy.

We then evaluated our algorithm through extensive simulations. On our lab equipment we found that the optimal algorithm fails due to insufficient memory when attempting to enumerate beyond rank 2^{33}, while our bounded-space algorithm continued its near-optimal-order enumeration unhindered.

We based our simulation on a new model of the ranking provided by an SCA, that may be of independent interest: We compared the empirical subkey distributions of an actual SCA with the true rank distribution, and discovered that the probabilities predicted by the SCA are overly optimistic. We found that the true rank distribution is long-tailed and is well modeled by a Pareto

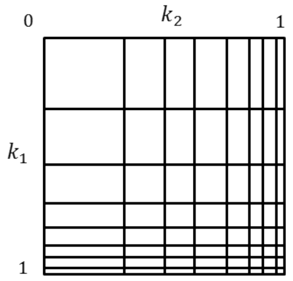

Fig. 1. Geometric representation of the key space

distribution, whose guessing entropy is significantly greater than that predicted by the SCA.

Organization: In Section 2 we describe the optimal-order key enumeration algorithm of [20]. In Section 3 we introduce some bounds on the guessing entropy of the full key based on the guessing entropies of the individual subkeys. In Section 4 we introduce our w-layer key enumeration algorithm and analyze its properties. In Section 5 we describe a new model of the subkey true rank probability distribution. In Section 6 we present our performance analysis, and we conclude in Section 7.

2 Preliminaries

The key enumeration problem: The cryptanalyst obtains d independent subkey spaces $k_1, ..., k_d$, each of size n, and their corresponding probability distributions $P_{k_1}, ..., P_{k_d}$. The problem is to enumerate the full-key space in deceasing probability order, from the most likely key to the least, when the probability of a full key is defined as the product of its subkey's probabilities.

The best key enumeration algorithm so far was presented by Veyrat-Charvillon, Gérard, Renauld and Standaert in [20]. To explain the algorithm, we will use a graphical representation of the key space—the case of $d = 2$ is depicted in Figure 1. In this figure, we see two subkeys k_1 and k_2 along the axes of the graph, both sorted by decreasing order of probability. The width and the height correspond to the probability of the corresponding subkey. Let $k_i^{(j)}$ denote the j'th likeliest value for the i'th subkey. Then, the intersection of row j_1 and column j_2 is a rectangle corresponding to the key $(k_1^{(j_1)}, k_2^{(j_2)})$ whose probability is equal to the area of the rectangle.

The algorithm outputs the keys in decreasing order of probability. The algorithm maintains a data structure F of candidates to be the next key in the sorted order. In each step the algorithm extracts the most likely candidate from F, $(k_1^{(j_1)}, k_2^{(j_2)})$, and outputs it. F is then updated by inserting the potential

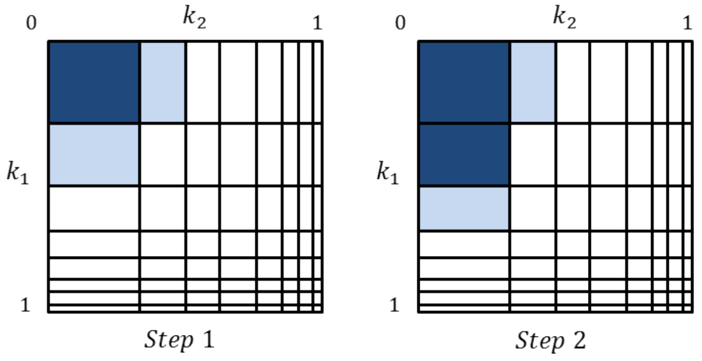

Fig. 2. Geometric representation of the first two steps of key enumeration.

successors of this candidate: $(k_1^{(j_1+1)}, k_2^{(j_2)})$ and $(k_1^{(j_1)}, k_2^{(j_2+1)})$. An important observation made by [20] is that F should never include 2 candidates in the same column, or in the same row: one candidate will clearly dominate the other. Thus the algorithm maintains auxiliary data structures ("bit vectors") to indicate which rows and columns currently have member in F. This observation has a crucial effect on the size of the data structure, $|F|$.

We can see in Figure 2 the first steps of the algorithm: the most likely key is $(k_1^{(1)}, k_2^{(1)})$, therefore this is the key that is output first (represented in dark gray in step 1). Now, the only possible next key candidates are the successors (represented in light gray in step 1) $(k_1^{(2)}, k_2^{(1)})$ and $(k_1^{(1)}, k_2^{(2)})$, which are inserted into F. Then again, the most likely key is extracted, but this time only one successor is inserted because there is already a key in column 2.

In general the number of dimensions $d > 2$ and we need to enumerate over more than two lists of subkeys. For AES, typically $d = 16$ for byte-level side channels or $d = 4$ for 32-bit subkeys as in [15]. To do this, [20] suggested a recursive decomposition of the problem. The algorithm described above is only used for merging two lists, and its outputs are used to form larger subkey lists which are in turn merged together. In order to minimize the storage and the enumeration effort, these lists are generated only as far as required by the key enumeration. Therefore, whenever a new subkey is inserted into the candidate set, its value is obtained by applying the enumeration algorithm to the lower level, (for example 64-bit subkeys obtained by merging two 32-bit subkeys), and so on.

3 Bounding the Guessing Entropy

An important security metric for the evaluation of a side channel attack [19] is the Guessing Entropy, which intuitively corresponds to the average number of keys to test before reaching the correct one, based on the probabilities assigned to key candidates by the side channel attack.

Definition 1 (Guessing Entropy). *For a random variable X with n values, denote the elements of its probability distribution P_X by $P_X(x_i)$ for $x_i \in X$ such that $P_X(x_1) \geq P_X(x_2) \geq ... \geq P_X(x_n)$. The guessing entropy of X is:*

$$G(X) = \sum_{i=1}^{n} i \cdot P_X(x_i).$$

The case $d = 2$: Let the key be split into 2 independent subkey spaces X and Y, each of size n, thus a key is a vector xy s.t. $x \in X$ and $y \in Y$. A side channel attack produces 2 separate distributions $P_X(x_i)$ for $x_i \in X$ and $P_Y(y_j)$ for $y_j \in Y$. Assume that the subkey distributions are sorted: $P_X(x_1) \geq P_X(x_2) \geq ... \geq P_X(x_n)$ and similarly for P_Y, then $G(X)$ and $G(Y)$ are well defined.

Let XY denote the list of (full) keys sorted in decreasing order of probability, where $P_{XY}(x_i, y_j) = P_X(x_i)P_Y(y_j)$ since the subkeys are independent. Thus $G(XY)$ is well defined. However, calculating $G(XY)$ requires a time and space complexity of $\Omega(n^2)$. Therefore bounding $G(XY)$ in terms of the easy-to-compute $G(X)$ and $G(Y)$ is a useful goal. To this end, let $rank(x_i, y_j)$ be the position of key (x_i, y_j) in XY. Clearly, $rank(x_1, y_1) = 1$ and $rank(x_n, y_n) = n^2$. By definition we get:

$$G(XY) = \sum_{i=1}^{n} \sum_{j=1}^{n} rank(x_i, y_j) \cdot P_X(x_i)P_Y(y_j). \tag{1}$$

Theorem 1. *The guessing entropy of XY, $G(XY)$, is bounded by:*

$$G(X)G(Y) \leq G(XY) \leq n(G(X) + G(Y)) - G(X)G(Y). \tag{2}$$

Proof. As mentioned in [21], and as depicted in Figure 3 given a key (x_i, y_j), all the key candidates with higher indexes in both indexes, have a lower probability and higher rank, and all the key candidates with lower indexes, have a lower rank.

Taking advantage of this fact, $rank(x_i, y_j)$ can be bounded as follows:

$$ij \leq rank(x_i, y_j) \leq n^2 - (n - i)(n - j). \tag{3}$$

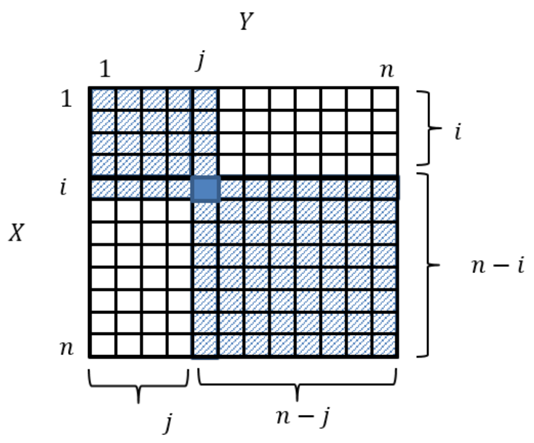

Fig. 3. Another geometric representation of the key space

We use this observation in order to bound $G(XY)$. First, we prove the lower bound. By substituting Eq. (3) in Eq. (1) we get:

$$\begin{aligned}
G(XY) &= \sum_{i=1}^{n}\sum_{j=1}^{n} rank(x_i, y_j) \cdot P_X(x_i)P_Y(y_j) \\
&\geq \sum_{i=1}^{n}\sum_{j=1}^{n} ij \cdot P_X(x_i)P_Y(y_j) \\
&= \sum_{i=1}^{n} i \cdot P_X(x_i) \sum_{j=1}^{n} j \cdot P_Y(y_j) \\
&= G(X)G(Y).
\end{aligned}$$

Second, we prove the upper bound of $G(XY)$. By Eq. (3),

$$\begin{aligned}
rank(x_i, y_j) &\leq n^2 - (n-i)(n-j) \\
&= n^2 - [n^2 - n(i+j) + ij] \\
&= n(i+j) - ij.
\end{aligned}$$

By substituting Eq. (3) in Eq. (1) we get:

$$\begin{aligned}
G(XY) &= \sum_{i=1}^{n}\sum_{j=1}^{n} rank(x_i, y_j) \cdot P_X(x_i)P_Y(y_j) \\
&\le \sum_{i=1}^{n}\sum_{j=1}^{n} \big(n(i+j) - ij\big) \cdot P_X(x_i)P_Y(y_i) \\
&= n\sum_{i=1}^{n}\sum_{j=1}^{n}(i+j) \cdot P_X(x_i)P_Y(y_j) - \sum_{i=1}^{n}\sum_{j=1}^{n} ij \cdot P_X(x_i)P_Y(y_j) \\
&= n\sum_{i=1}^{n}\sum_{j=1}^{n}(i+j) \cdot P_X(x_i)P_Y(y_j) - G(X)G(Y). \qquad (*)
\end{aligned}$$

We calculate separately the summation at Eq. (*),

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n}(i+j) \cdot P_X(x_i)P_Y(y_j) &= \sum_{i=1}^{n} P_X(x_i) \sum_{j=1}^{n}(i+j) \cdot P_Y(y_j) \\
&= \sum_{i=1}^{n} P_X(x_i) \Big[\sum_{j=1}^{n} i \cdot P_Y(y_i) + \sum_{j=1}^{n} j \cdot P_Y(y_j)\Big] \\
&= \sum_{i=1}^{n} i \cdot P_X(x_i) \underbrace{\sum_{j=1}^{n} P_Y(y_j)}_{1} \\
&+ \sum_{j=1}^{n} P_X(x_i) \underbrace{\sum_{j=1}^{n} j \cdot P_Y(y_i)}_{G(Y)} \\
&= G(X) + G(Y).
\end{aligned}$$

By substituting the results in Eq. (*) we get:

$$G(XY) = n\big(G(X) + G(Y)\big) - G(X)G(Y) \qquad (4)$$

Which concludes the proof of the theorem. We can see that in general $G(XY)$ is not multiplicative:

Corollary 1 $G(X)G(Y) \le G(XY) \le 2n \cdot \max\big(G(X), G(Y)\big)$.

Proof. Without loss of generality, we assume that $G(X) \ge G(Y) \ge 1$ then Eq. (4) can be bounded by:

$$G(XY) \le n \cdot 2G(X) - G(X) = (2n-1)G(X) \le 2n \cdot \max\big(G(X), G(Y)\big).$$

These bounds can be expanded for $d > 2$. In this case it holds:

$$\prod_{m=1}^{d} i_m \le rank(x_{i_1}^{(1)}, x_{i_2}^{(2)}, ..., x_{i_d}^{(d)}) \le n^d - \prod_{m=1}^{d}(n - i_m).$$

Therefore we obtain

Theorem 2. *The guessing entropy $G(X^{(1)}X^{(2)}...X^{(d)})$, is bounded by:*

$$\prod_{m=1}^{d} G(X^{(m)}) \leq G(X^{(1)}X^{(2)}...X^{(d)}) \leq n^d - \prod_{m=1}^{d} (n - G(X^{(m)})).$$

As an example of using these bounds, with byte-level SCA on AES we have $d = 16$. If the SCA discards 128 values per byte and returns a probability distribution over the remaining 128 candidates we have $n = 128$. Assuming that $G(X^{(m)}) = 8$ for all 16 subkeys we get that

$$2^{48} = 8^{16} \leq G(X^{(1)}X^{(2)}...X^{(d)}) \leq 128^{16} - (128 - 8)^{16} = 2^{111.36}.$$

Note that the upper bound is rather weak, especially when $G(X)$ is low—which is usually the case in successful SCA scenarios.

4 Key Enumeration algorithm

The key enumeration in [20] enumerates the key candidates in optimal order, but has a significant drawback, its memory requirements may exceed the available memory. Its worst-case space complexity is $\Omega(n^{d/2})$ since it needs to store the full sorted distribution of the 2 top-level dimensions (in addition to the data structure F). Moreover, in order to enumerate until a key of rank $r \leq n^{d/2}$ it has a space complexity of $\Omega(r)$. In this section, we present a new key enumeration algorithm with bounded memory requirements, which therefore allows to enumerate a large number of key candidates.

To achieve the desired memory bound, we relax the "optimal order" requirement: our algorithm enumerates the keys in near-optimal order, and we are able to bound the ratio between the optimal rank of a key and our algorithm's rank of that key.

4.1 The layering approach

In order to explain our algorithm, we start with the case $d = 2$. We divide the key-space into layers of width w, as depicted in Figure 4. The first layer contains the keys $(k_1^{(i)}, k_2^{(j)})$ such that $(i, j) \in \{1, ..., w\} \times \{1, ..., w\}$. The second layer contains the keys $(k_1^{(i)}, k_2^{(j)})$ such that $(i, j) \in \{1, ..., 2w\} \times \{1, ..., 2w\} \backslash \{1, ..., w\} \times \{1, ..., w\}$ and so on. More formally:

Definition 2. *Given $w > 0$ and $l > 0$, let*

$$layer_l^w = \{(k_1^{(i)}, k_2^{(j)}) | (i, j) \in \{1, ..., l \cdot w\} \times \{1, ..., l \cdot w\} \backslash \{1, ..., (l-1) \cdot w\} \times \{1, ..., (l-1) \cdot w\}\}.$$

A key observation is that we can run the optimal enumeration algorithm of [20] *within* a layer: we seed the algorithm data structure F by inserting the two "corners" (see Figure 4), and then extract candidates and insert their successors as usual - limiting ourselves not to exceed the boundaries of the layer. Moreover, within a layer of width w, we can bound the space used by F:

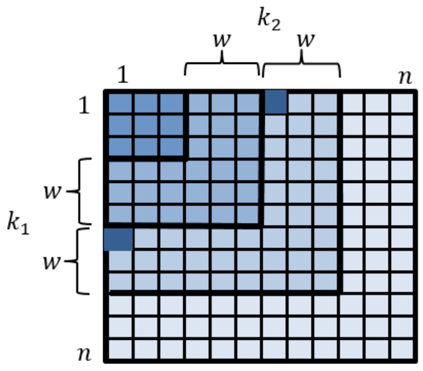

Fig. 4. Geometric representation of the key space divided into layers of width $w = 3$. The keys in cells $(1,7)$ and $(7,1)$ are the algorithm's seeds for $layer_3^{(3)}$

Proposition 1. *For every $l > 0$ and $w > 0$, applying the optimal key enumeration of [20] on $layer_l^w$, the number of next potential key candidates is bounded by $2w$, i.e., $|F| \leq 2w$.*

Proof. The proof is directly derived from the key enumeration algorithm [20]. The algorithm extracts the highest probability key candidate from the data structure F, and inserts its successors into F, while keeping the rule that F may contain at most one element in each row and column of $layer_l^w$. Looking at the geometrical representation, $layer_l^w$ is a union of a horizontal rectangle H_l^w and a vertical rectangle V_l^w, such that

$$H_l^w = \{(k_1^{(i)}, k_2^{(j)}) \in layer_l^w | (l-1) \cdot w < i \leq l \cdot w\},$$

$$V_l^w = \{(k_1^{(i)}, k_2^{(j)}) \in layer_l^w | (l-1) \cdot w < j \leq l \cdot w\}.$$

$$layer_l^w = H_l^w \cup V_l^w.$$

Therefore, the size of F, for every $layer_l^w$ is:

$$|F| \leq |F \cap H_l^w| + |F \cap V_l^w|$$

Because F may contain at most one element in each column and row, the size of $|F \cap H_l^w|$ is the minimum between H_l^w's two dimensions:

$$|F \cap H_l^w| \leq min\{w, l \cdot w\} = w,$$

and similarly for $|F \cap V_l^w|$. Therefore, we get

$$|F| \leq 2w.$$

Importantly, the bound on $|F|$ is independent of n, and depends only on the design parameter w which we can tune.

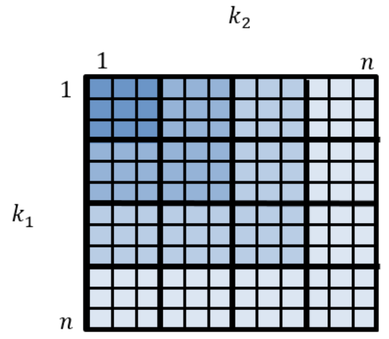

Fig. 5. Geometric representation of the key space divided into squares of width $w = 3$.

4.2 The Two-Dimensional Algorithm

Proposition 1 leads us to our *w-layer key enumeration* algorithm: Divide the key-space into layers of width w. Then, go over the $layer^w$s, one by one, in increasing order. For each $layer_l^w$, enumerate its key candidates, by applying the optimal key enumeration [20]. Following the proposition, the number of potential next candidates, F, that our algorithm should store is bounded by $2w$.

For $d = 2$, the sorted orders of keys in the 2 subkey spaces are given explicitly. However, for $d > 2$ during the recursive descent, we need to generate the ordered lists on the fly as far as required. We do this by applying a recursive decomposition of the problem. The length of these generated subkey spaces grows and eventually becomes too large to store. Therefore, instead of naively storing the full subkey order, we only store the $O(w)$ candidates which were computed recently.

To do this, we divide each $layer^w$ in the geometrical representation, into squares of size $w \times w$, as depicted in Figure 5. Our algorithm still enumerates the key candidates in $layer_1^w$ first, then in $layer_2^w$ and so on, but in each $layer_l^w$ the enumeration will be square-by-square.

More specifically, let $S_{x,y}^w$ be a set of the key candidates in the square $S_{x,y}^w = \{(k_1^{(i)}, k_2^{(j)}) | (x-1) \cdot w < i \leq x \cdot w \text{ and } (y-1) \cdot w < j \leq y \cdot w\}$. We say that two squares, $S_{x,y}$ and $S_{z,w}$ are *in the same row* if $y = w$, and are *in the same column* if $x = z$.

Now let's describe the enumeration at each $layer_l^w$. We know that the most likely candidate in $layer_l^w$ is either at $S_{1,l}$ or $S_{l,1}$. Therefore, we enumerate first the key candidates in $S_{1,l} \cup S_{l,1}$ by applying the key enumeration in [20] on them (represented in dark gray in step 1 in Figure 6). Let S denote the set of squares that contain potential next candidates in this layer. At some point, one of the two squares is completely enumerated. Without loss of generality, we assume this is $S_{1,l}$. At this point, the only square that contains the next key candidates after $S_{1,l}$ is the successor $S_{2,l}$ (represented in dark gray in step 2 Figure 6).

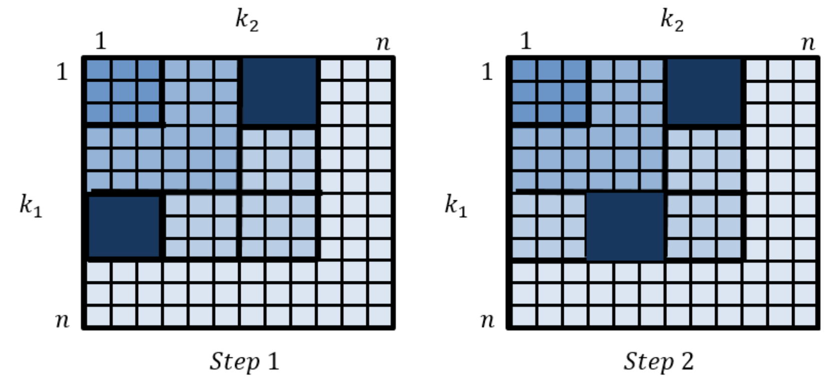

Fig. 6. Geometric representation of the key enumeration at $layer_3^3$.

In general case, the successor of $S_{x,y}$ is either $S_{x+1,y}$ or $S_{x,y+1}$, only one of which is in $layer_l^w$. Therefore, when one of the squares is completely enumerated, it is extracted from S, and its successor is inserted, as long as S doesn't contain a square in the same row or column.

Notice that only after a square is completed we continue to it's successor. Without loose of generality, we assume that the successor is in the same row as the current one. Therefore, for all candidates $(k_1^{(i)}, k_2^{(j)})$ we intend to check next, the j index is higher than the j index of any candidate in the current square, therefore these j indexes of the current square are useless hence are not stored.

It is simple to see that S contains at most 2 squares of size $w \times w$ each step, therefore only the subkeys at each dimension should be stored, which means a space complexity of $O(w)$.

4.3 Generalization to multi-dimensional Algorithm

Similarly to [20] we apply a recursive decomposition of the problem. Whenever a new subkey is inserted into the candidate set, its value is obtained by applying the enumeration algorithm to the lower level, (for example 64-bit subkeys obtained by merging two 32-bit subkeys). For example, let's look at $d = 4$. In order to generate the ordered full-key, we need to generate the 2 ordered lists of the lower level on the fly as far as required. In the worst case the space complexity of these 2 lists is $\Omega(n^2)$. Although the size of the data structure F is bounded by $2w$, we still have a bottleneck of $\Omega(n^2)$. Therefore, instead of naively storing the full subkey order, we only store the $O(w)$ candidates which computed recently. We store $2w$ subkeys of the first low-level list, and another $2w$ subkeys of the second low-level list.

Algorithm 1: w-Layer Key Enumeration Algorithm.

```
Input: Subkey distributions $\{k_i\}_{1\leq i\leq d}$.
Output: The correct key, if exists, NOT-FOUND otherwise.
1  $currentLayer = 1$;
2  $found = false$;
3  while (currentLayer is not out of range) do
4      $S \leftarrow S_{1,currentLayer} \cup S_{currentLayer,1}$;
5      while ($S \neq \emptyset$) do
6          $candidate$ = nextCandidate($S, \{k_i\}_{1\leq i\leq d}$);
7          $found$ = isCorrectKey($candidate$);
8          if ($found$) then
9              return $candidate$;
10         end
11     end
12     $currentLayer{+}{+}$
13 end
14 return NOT-FOUND;
```

4.4 Bounding the Rank and the Guessing Entropy

Let v^w denote the vector resulting from enumerating all key candidates, applying our w-layer key enumeration, for fixed w, and let v denote the vector resulting from applying the optimal order enumeration. Additionally, let $rank^w(i_1, i_2, .., i_d)$ denote the order statistic of key $(k_1^{(i_1)}, k_2^{(i_2)}, ..., k_d^{(i_d)})$ in v^w, and $rank(i_1, i_2, .., i_d)$ be the order statistic of key $(k_1^{(i_1)}, k_2^{(i_2)}, ..., k_d^{(i_d)})$ in v . Now, we want to bound the rank of the w-layer algorithm, and the guessing entropy of v^w, $G(v^w)$, related to $G(v)$.

Theorem 3. *Consider a key $(k_1^{(i_1)}, ..., k_d^{(i_d)})$. Let $i^* = \max\{i_1, ..., i_d\}$, and let $\alpha_m = i_m/i^*$ for $m = 1, ..., d$ ($\alpha_m \leq 1$). Then,*

$$rank^w(i_1, ..., i_d) \leq \prod_{m=1}^{d} \left(\frac{2}{\alpha_m}\right) \cdot rank(i_1, ..., 1_d).$$

Proof. According Eq. (3), it holds that:

$$rank(i_1, i_2, ..., i_d) \geq \prod_{m=1}^{d} i_m.$$

Without loss of generality, we assume that the maximal index $i^* = i_1$. Using that, we get:

$$rank(i_1, i_2, ..., i_d) \geq \prod_{m=1}^{d} i_m = i_1^d \cdot \prod_{m=1}^{d} \alpha_m. \tag{5}$$

Our goal is to upper-bound $rank^w(i_1, i_2, .., i_d)$. We divide the analysis into two cases based on the value of i_1.

Algorithm 2: nextCandidate.

```
Input: Set of squares with potential candidates $S$ and Subkey distributions
       $\{k_i\}_{1 \le i \le d}$.
Output: The next key candidat in $S$.
1  $main \triangleq \{1, ..., d\}$;
2  $x \triangleq \{1, ..., d/2\}$;
3  $y \triangleq \{d/2+1, ..., d\}$;
4  $(k_x^{(i)}, k_y^{(j)}) \leftarrow$ most likely candidate in $F_{main}$;
5  $F_{1,d} \leftarrow F_{1,d} \setminus \{(k_x^{(i)}, k_y^{(j)})\}$;
6  $S_{i,j} \leftarrow$ the square which $(k_x^{(i)}, k_y^{(j)})$ is inside;
7  if $S_{i,j}$ is completely enumerated then
8      $S \leftarrow S \setminus S_{i,j}$;
9      if no square in same row/column as $Successor(S_{i,j})$ then
10         $S \leftarrow S \cup Successor(S_{i,j})$;
11         $F_{main} \leftarrow F_{main} \cup \{$most likely candidate in $Successor(S_{i,j})\}$;
12     end
13 else
14     if $(k_x^{(i+1)}, k_y^{(j)}) \in S$ and no candidate in row i+1 then
15         if $k_x^{(i+1)}$ hasn't calculated yet then
16             nextCandidate($\{k_i\}_{i \in x}$);
17             $k_x^{(i+1)} \leftarrow$ the most likely candidate in $F_x$ ;
18         end
19         $F_{main} \leftarrow F_{main} \cup \{(k_x^{(i+1)}, k_y^{(j)})\}$;
20     end
21     if $(k_x^{(i)}, k_y^{(j+1)}) \in S$ and no candidate in column j+1 then
22         if $k_y^{(j+1)}$ hasn't calculated yet then
23             nextCandidate($\{k_i\}_{i \in x}$);
24             $k_y^{(j+1)} \leftarrow$ the most likely candidate in $F_y$ ;
25         end
26         $F_{main} \leftarrow F_{main} \cup \{(k_x^{(i)}, k_y^{(j+1)})\}$;
27     end
28 end
29 return $(k_x^{(i)}, k_y^{(j)})$ ;
```

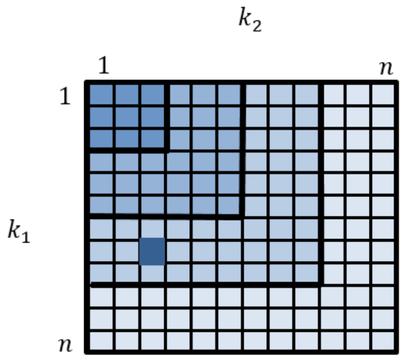

Fig. 7. Geometric representation of key $(k_1^{(i_1)}, k_2^{(i_2)})$ at $layer_2^3$ with $i_1 = 8$, $i_2 = 3$ and $w = 3$.

case 1: $i_1 \geq w$. In this case, the key $(k_1^{(i_1)}, k_2^{(i_2)}, ..., k_d^{(i_d)})$ is at some $layer_l^w$ such that $l > 1$. According to our w-layer key enumeration, we enumerate all the keys in the current layer, before we continue to the next layer. Therefore, the rank of this key is bounded by the total number of keys in layers $1, .., l$. As we can see in Figure 7, for the bi-dimensional case, the distance between i_1 and the high bound of the layer l is at most w. Since i_1 was maximal, the high bound of the layer l is at most $i_1 + w$ for all d dimensions, and $rank^w(i_1, i_2, ..., i_d)$ is upper-bound by

$$rank^w(i_1, i_2, ..., i_d) \leq (i_1 + w)^d. \tag{6}$$

Using our assumption that $i_1 \geq w$, we get:

$$(i_1 + w)^d \leq (2 \cdot i_1)^d = 2^d \cdot i_1^d. \tag{7}$$

Plugging Eq. (6) and Eq. (7) into Eq. (5) yields

$$rank^w(i_1, i_2, ..., i_d) \leq \prod_{m=1}^{d} \left(\frac{2}{\alpha_m}\right) \cdot rank(i_1, i_2, ..., i_d).$$

case 2: $i_1 < w$. In this case, the key $(k_1^{(i_1)}, k_2^{(i_2)}, ..., k_d^{(i_d)})$ is at the first layer $layer_1^w$. Therefore, our w-layer key enumeration, enumerates the keys in the same order the optimal key enumeration does, possibly skipping the keys which are outside this layer. Therefore:

$$rank^w(i_1, i_2, ..., i_d) \leq rank(i_1, i_2, ..., i_d)$$

and in particular,

$$rank^w(i_1, i_2, ..., i_d) \leq \prod_{m=1}^{d} \left(\frac{2}{\alpha_m}\right) \cdot rank(i_1, i_2, ..., i_d).$$

since $\frac{2}{\alpha_m} > 1$ for all m.

Theorem 4. *The boundary of the guessing entropy of v^w, $G(v^w)$, related to $G(v)$ is:*

$$G(v^w) \leq 2^d n^{d-1} \cdot G(v).$$

Proof. Each subkey contains n values, therefore, i_1 is at most n times bigger than any of $i_2, ..., i_d$, means for each $m \in \{2, ..., d\}$, $\alpha_m \geq 1/n$. Applying this on Theorem 3, we get:

$$rank^w(i_1, i_2, ..., i_d) \leq 2^d \cdot n^{d-1} \cdot rank(i_1, i_2, ..., i_d).$$

Therefore,

$$G(v^w) \leq 2^d \cdot n^{d-1} \cdot G(v).$$

4.5 Space Complexity Analysis

The algorithm needs to store for each level a list, of size w, of subkeys obtained from applying the algorithm on lower level. For this, it needs to store 2 squares of size $w \times w$ for the 2 top-level dimensions, which means 4 lists. In addition, it needs to store, for the current level, a data structure F which bounded by $2w$ and 2 data structures ("bit vectors") to indicate which rows and columns currently have member in F. Totally we get the following space recurrence relation:

$$S(d) = 4S(d/2) + cw,$$

for some constant c, which sums to $O(d^2 w)$. Taking into account the input, whose space is $O(dn)$, we get a total space complexity of $O(d^2 w + dn)$.

5 Modeling the subkey probability distribution

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. However, with data sets such as the DPAv4 [1] we know the true correct keys over traces from many different keys. Hence, it is possible to compare to SCA-predicted probability distribution with reality.

Assuming that the SCA-distribution P_X^k for a subkey X on key instance k, is sorted in decreasing order, hence $P_X^k(x_i)$ is the SCA probability that the correct key of instance k has rank i. However, we also know the true rank r^* of the correct key for each instance k, according to the order implied by P_X^k. Thus we can calculate the true rank probability distribution: let $P_X^*(r)$ be the frequency of subkey instances such that the subkey at position r according to the SCA-order is the correct subkey.

Figure 8 shows the SCA-predicted probability distribution versus the true rank distribution, using the data of Oren, Weisse and Wool in [15].

The figure shows that the SCA distribution is too optimistic: the cdf grows much faster than that of P_X^*. This also leads to a significant difference in the

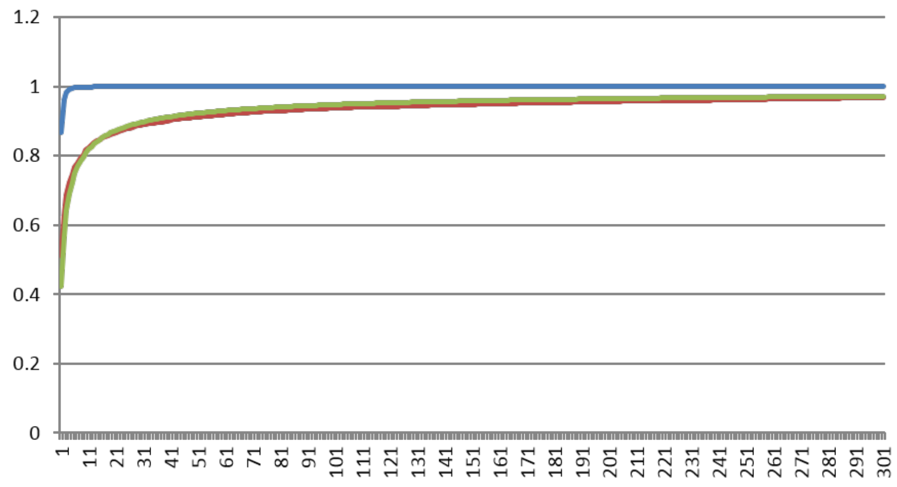

Fig. 8. The cdf of the average SCA-predicted probability distribution (top curve - blue), the true empirical rank probability distribution P^*_X (red) and the synthetic Pareto distribution (green) as function of the rank. Note that the cdf curves for the empirical and synthetic distributions overlap almost completely.

guessing entropy, with $G(X) = 1.2$ to the SCA-predicted compared to 135 for the true rank probability distribution.

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, then the PDF is given by:

$$f_X(x) = \frac{\alpha}{x^{\alpha+1}}, \quad \text{for } x \geq 1.$$

In order to simulate the true rank distribution, we choose parameters that lead the expected value to equal the guessing entropy of the rank distribution, 135, by solving the following equations:

$$\beta \sum_{x=1}^{n} f_X(x) = \beta \cdot \sum_{i=1}^{n} \frac{\alpha}{x^{\alpha+1}} = 1$$

and

$$E(X) = \beta \sum_{x=1}^{n} x \cdot f_X(x) = \beta \sum_{x=1}^{n} \frac{\alpha}{x^{\alpha}} = 135.$$

For $d = 4$ and $n = 2^{17}$ (as in the results of [15]), we obtain $\alpha = 0.575$, and $\beta = 0.738$. Figure 8 also shows a curve for this Pareto distribution demonstrating the fit.

This observation leads to two conclusions. First, it shows that, at least for the method of [15], the SCA-predicted distribution and the claimed guessing entropy are overly optimistic - it would be interesting to evaluate other SCA attacks using this methodology. Second, using Pareto distributions as a model we can simulate various scenarios for the key enumeration algorithms without creating an actual SCA attack.

6 Performance Analysis

We evaluated the performance of our w-layer key enumeration algorithm through an extensive simulation study. We implemented the optimal algorithm [20] and our algorithm in Java, and ran both algorithms on a 3.07GHz PC with 24GB RAM running Microsoft windows 7, 64bit. Note that the code of the optimal algorithm is used as a subroutine in the w-layer algorithm, thus any potential improvement in the former's implementation would automatically translate into an analogous improvement in the latter.

We used synthetic SCA distributions with $d = 8$ dimensions and $n = 2^{12}$ for a total enumeration space of 2^{96}. The 8 probability distributions were generated according to the model of Section 5: Pareto distributions fitting $G(X) = 135$, with $\alpha = 0.575$ and $\beta = 0.738$. We analyzed our w-layer algorithm for two different values of w: (i) $w = n = 2^{12}$ and (ii) $w = 2^{25}$.

We also evaluated the algorithm's performance for $d = 16$ dimensions and $n = 2^6$, again for a total enumeration space of 2^{96}. The 16 probability distributions were Pareto distributions fitting $G(X) = 8$, with $\alpha = 0.3$ and $\beta = 1.1197$. We analyzed our w-layer algorithm for two different values of w: (i) $w = n = 2^6$ and (ii) $w = 2^{25}$. The obtained results are similar to those with $d = 8$. Graphs are omitted.

We conducted the experiments as follows. We ran the optimal algorithm on different (optimal) ranks starting from 2^{12}, and measured its time and space consumption. For each optimal rank, 2^x, we extracted the key corresponding to this rank, and ran each of our w-layer key enumeration algorithm variants until it reached the same key, and measured its rank, time and space. We repeated this simulation for 64 different ranks near 2^x — the graphs below display the median of the measured values.

6.1 Runtime Analysis

Figure 9 illustrates the time (in minutes) of the 3 algorithms: optimal-order (green triangles), w-layer with $w = n$ (red squares) and w-layer with $w = 2^{25}$ (blue diamonds) for different ranks. The figure shows that, crucially, the optimal-order key enumeration stops at 2^{33}. This is because of high memory consumption which exceeds the available memory.

For ranks beyond 2^{22} we noticed that the w-layer enumeration with $w = n = 2^{12}$ became significantly slower than the others. The red squares in Figure 9 are misleadingly low, since as Figure 10 shows, a large fraction of runs timed-out

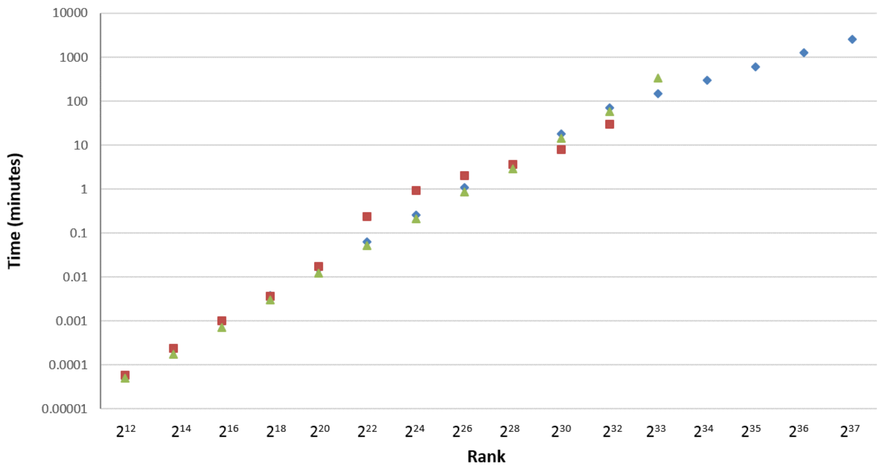

Fig. 9. Time, in minutes, of optimal-order key enumeration (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks.

at the 2 hour mark, and we stopped experimenting with this setting beyond rank 2^{32}. It is important to remark that we chose to stop because of the time consumption - the algorithm doesn't stop till it gets to the correct key.

For the w-layer with $w = 2^{25}$ we can see excellent results. For small ranks it takes exactly the same time consumption as the optimal-order, (hidden by the green triangles in Figure 9), and for high ranks, its bounded space complexity enables it to enumerate in reasonable time.

Note that for ranks beyond 2^{33}, the optimal algorithm failed to run, so we could not identify the keys with those ranks. In order ro demonstrate the w-layer algorithm's ability to continue its enumeration we let it run until it reached a rank r in its own near-optimal order (for $r = 2^{34}, .., 2^{37}$) - and for those experiments we removed the 2 hour time out.

6.2 Space Utilization

Figure 11 illustrates the space (in bytes) used by the 3 algorithms' data structures for different ranks. As we can see again, the optimal-order key enumeration stops at 2^{33} because memory shortage. For the w-layer algorithm with $w = n$ we can clearly see the bounded space consumption leveling at around 1MB. For the w-layer algorithm with $w = 2^{25}$ we see that its space consumption levels around 4GB and remains steady - allowing the algorithm to enumerate further into the key space, limited only by the time the cryptanalysis is willing to spend.

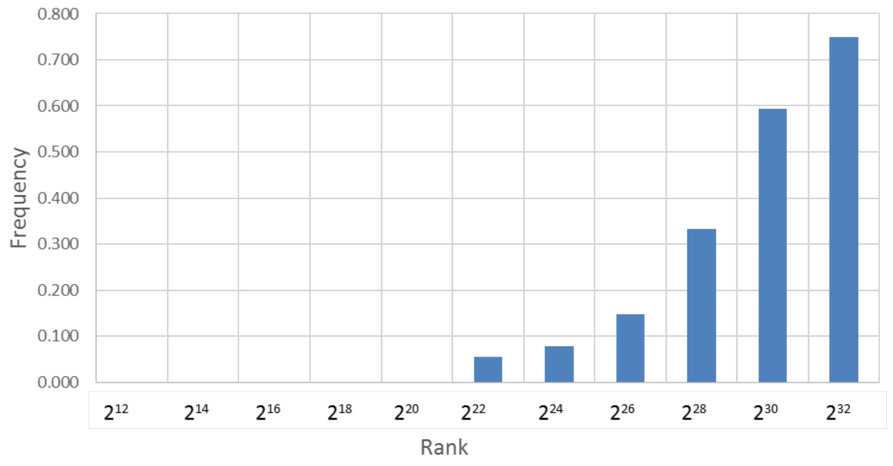

Fig. 10. Frequency of the keys whose time consumption applying the w-layer key enumeration with $w = n$ is higher than 2 hours.

6.3 The Difference in Ranks

Figure 12 illustrates the ranks detected by the 3 algorithms as a function of the optimal rank. By definition the optimal algorithm finds the correct ranks. Despite the somewhat pessimistic bounds of Theorem 4, the figure shows that with $w = n$ the ratio between the optimal rank are $rank^w$ is approximately 5 (again, beyond 2^{28} too many runs timed out for meaningful data). For $w = 2^{25}$ the discovered ranks are almost identical to the optimal ranks (the symbols in the figure overlap) - and beyond 2^{33} the optimal algorithm failed so comparison is not possible.

7 Conclusion

In this paper, we investigated the side channel attack improvement obtained by adversaries with non-negligible computation power to exploit physical leakage. For this purpose, we presented a new w-layer key enumeration algorithm that trades-off the optimal enumeration order in favor of a bounded memory consumption. We analyzed the algorithm's space complexity, guessing entropy, and rank distribution. We also evaluated its performance by extensive simulations. As our simulations show, our w-layer key enumeration allows stronger attacks than the order-optimal key enumeration [20], whose space complexity grows quickly with the rank of the searched key—and exceeds the available RAM in realistic scenarios. Since our algorithm can be configured to use as much RAM as available (but no more) it can continue its near optimal enumeration unhindered.

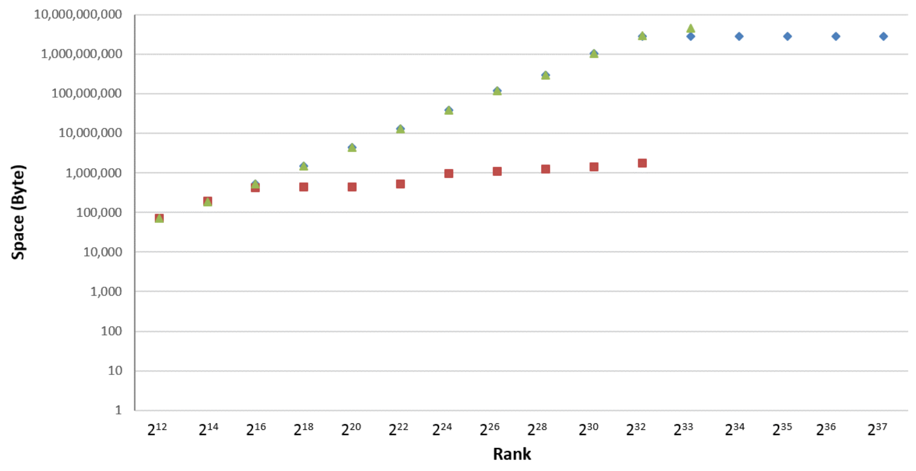

Fig. 11. Space, counting the data structure elements, of optimal-order key enumeration (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks.

Along the way we provided bounds on the full key guessing entropy in terms of the guessing entropies of the individual subkeys. We also observed a significant gap between the predicted and the true rank distributions of a real SCA—we showed that the true rank distribution is long-tailed, and well modeled by a Pareto distribution.

Finally, an open-source Java implementation for both our w-layer key enumeration and the order-optimal enumeration [20] are available via the authors' home pages.

References

1. DPA contest v4. `http://www.dpacontest.org/v4/`.
2. Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. The EM side-channel (s). In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 29–45. Springer, 2003.
3. Barry C Arnold. *Pareto distribution*. Wiley Online Library, 1985.
4. Daniel J Bernstein, Tanja Lange, and Christine van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. Cryptology ePrint Archive, Report 2015/221, 2015. `http://eprint.iacr.org/`.
5. 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.

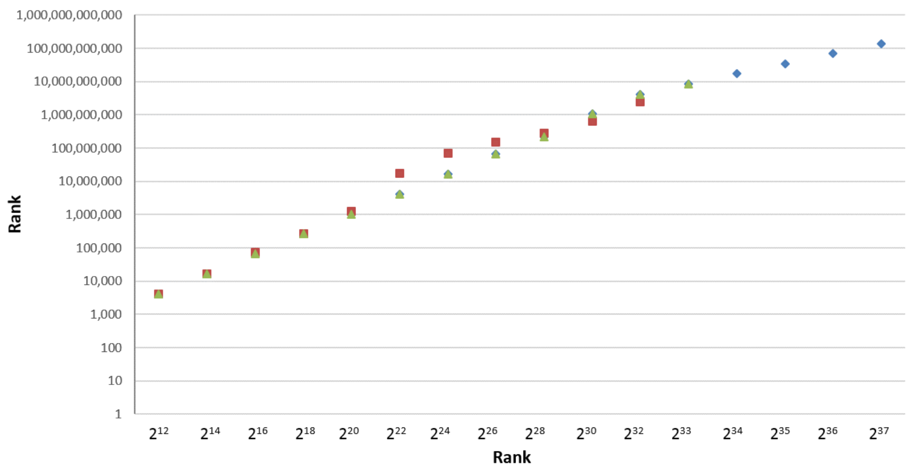

Fig. 12. Rank of optimal-order key enumeration (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks.

6. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 16–29. Springer, 2004.
7. Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 13–28. Springer, 2003.
8. Markus Dichtl. A new method of black box power analysis and a fast algorithm for optimal key search. *Journal of Cryptographic Engineering*, 1(4):255–264, 2011.
9. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems—CHES 2001*, pages 251–261. Springer, 2001.
10. Pascal Junod and Serge Vaudenay. Optimal key ranking procedures in a statistical cryptanalysis. In *Fast Software Encryption*, pages 235–246. Springer, 2003.
11. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer, 1999.
12. Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer, 1996.
13. 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.
14. Willi Meier and Othmar Staffelbach. Analysis of pseudo random sequences generated by cellular automata. In *Advances in Cryptology—EUROCRYPT'91*, pages 186–199. Springer, 1991.

15. 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.
16. 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.
17. Romain Poussier, Vincent Grosso, and François-Xavier Standaert. Comparing approaches to rank estimation for side-channel security evaluations. In *Smart Card Research and Advanced Applications (CARDIS)*. Springer, 2015.
18. 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.
19. 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.
20. Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography (SAC)*, pages 390–406. Springer, 2013.
21. 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.
22. 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. Springer, 2014.