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

Liron David(✉) and Avishai Wool(✉)

School of Electrical Engineering, Tel Aviv University, 69978 Ramat Aviv, Israel
lirondavid@gmail.com, 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. The best optimal-order key enumeration algorithms have a huge space complexity of $\Omega(n^{d/2})$ when there are d subkeys and n candidate values per subkey. In this paper, we present a parallelizable algorithm that enumerates the keys in near-optimal order but enjoys a much better space complexity of $O(d^2w + dn)$ for a design parameter w which can be tuned to available RAM.

Before presenting our algorithm, we provide lower and upper 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. Finally, we evaluate our algorithm through extensive simulations, to show the advantages of our new algorithm in practice, on realistic SCA scenarios. 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.

1 Introduction

1.1 Background

Side-channel attacks (SCA) represent a serious threat to the security of cryptographic hardware products. As such, they reveal the secret key of a cryptosystem based on leakage information gained from physical implementation of the cryptosystem on different devices. Information provided by sources such as timing [13], power consumption [12], electromagnetic emulation [20], electromagnetic radiation [1,9] 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.

H. Handschuh (Ed.): CT-RSA 2017, LNCS 10159, pp. 311–327, 2017.
DOI: 10.1007/978-3-319-52153-4_18

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) [12], Brier et al.'s Correlation Power Analysis (CPA) [5] and Chari et al.'s Template Attacks [6] 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 [11]. 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 [15]. In this work the attacker has no access to the subkey distributions but is able to generate subkeys according to them. The proposed solution is to enumerate keys by randomly choosing subkeys according to these distributions. This implementation requires $O(1)$ memory but keys may be chosen many times, leading to useless repetitions.

A deterministic enumeration algorithm was described by Pan et al. [17]. 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.

The currently best optimal algorithm was proposed by Veyrat-Charvillon, Gérard, Renauld and Standaert, [22], 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.

To tackle this problem, two improved key enumeration algorithms were proposed by Bogdanov et al. [4] and Martin et al. [14]. Similar to us, both papers improve upon OKEA [22] by suggesting bounded-memory algorithms.

Bogdanov et al. [4] 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, and can be parallelized. The algorithm of Martin et al. [14] also uses a score-based enumeration, focuses on rank estimation via a reduction to counting knapsack and utilizes it to enumerate the B keys with the highest scores in a parallel manner, for any B. Like [4] they also manipulate the side-channel leakages, but into different weights. Both use additive scoring (the scores of different subkeys are added to score a full key): [4] suggests scores that are scaled-and-truncated probabilities, whereas [14] skirts this issue. This makes it difficult to compare apples

with 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 both its order quality and its runtime, whereas their algorithms do not enjoy this benefit.

The most similar work to ours was developed in parallel to our technical report [7] by Poussier et al. [19]. The authors use a very different, histogram-based method, to enumerate the keys in parallelizable sub-optimal order. Like us they also use probabilities (technically, log-probabilities). Using our notation, their algorithm has a $\Omega(d^2 N_b + nd)$ space complexity—when N_b (number of bins) is a design parameter, i.e., the same asymptotic space complexity as our method. However, like both [4,14] Poussier et al. [19] did not provide any analytical bounds on the distance between their rank and the optimum, nor did they analyze the guessing entropy of their algorithm—they only provide empirical evidence based on one dataset.

Ye et al. [24] take a different approach: they limit the key enumeration to a hypercube of the top e candidates for every subkey. 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. [18] is primarily a taxonomy and comparison of rank estimation algorithms, suggesting new algorithmic combinations. It continues the work of Veyrat [23], Bernstein [3], Glowacz [10] and also of Martin et al. [14]. Rank estimation is a closely related, yet different, question, to the key enumeration we address: It doesn't necessarily require to enumerate all the key candidates ranked before the correct key, as it is only necessary to estimate how many there are.

1.3 Contributions

In this paper, we propose a parallelizable key enumeration algorithm, with bounded memory requirement of $O(d^2 w + dn)$ for a design parameter w which can be tuned to available RAM and allows the enumeration of a large number of keys without exceeding the available memory. Our algorithm enumerates in near-optimal order with a bounded ratio between optimal and near-optimal ranks.

Before presenting our algorithm, we utilize the evaluation framework of [21], providing lower and upper 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.

Finally, we evaluate our algorithm through extensive simulations, to show the advantages of our new algorithm in practice, on realistic SCA scenarios. On our lab environment 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.

Organization: In Sect. 2 we describe the optimal-order key enumeration algorithm of [22]. In Sect. 3 we introduce some bounds on the guessing entropy of the full key based on the guessing entropies of the individual subkeys. In Sect. 4 we introduce our w-layer key enumeration algorithm and analyze its properties. In Sect. 5 we present our performance analysis, and we conclude in Sect. 6.

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 decreasing 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, in terms of optimal-order, was presented by Veyrat-Charvillon, Gérard, Renauld and Standaert in [22], which we denote by OKEA. To explain the algorithm, we will use a graphical representation of the key space—the case of $d = 2$ is depicted in Fig. 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 of the rows and columns 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.

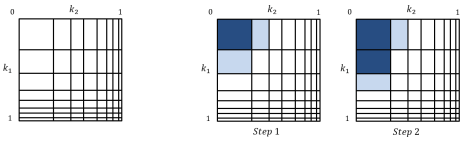

Fig. 1. Left: geometric representation of the key space. Right: geometric representation of the first two steps of key enumeration.

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 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 [22] 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 a member in F. This observation has a crucial effect on the size of the data structure, $|F|$.

We can see in Fig. 1 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 in step 2, 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, we need to enumerate over more than two lists of subkeys ($d > 2$). For AES, typically $d = 16$ for byte-level side channels or $d = 4$ for 32-bit subkeys as in [16]. To do this, [22] 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 [21] 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. Appears in the extended version of this paper [7].

We can see that in general $G(XY)$ is not multiplicative:

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

Proof. Appears in the extended version of this paper [7].

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

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

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

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

Reducing the gap between the lower and the upper bounds is left as an open question.

4 The Key Enumeration Algorithm

The key enumeration in [22] 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), for each dimension. 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 of two dimensions, $d = 2$. We divide the key-space $(n \times n)$ into layers of width w, as depicted in Fig. 2. 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 $(i, j) \in \{1, ..., 2w\} \times \{1, ..., 2w\} \setminus \{1, ..., w\} \times \{1, ..., w\}$ and so on. More formally:

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

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

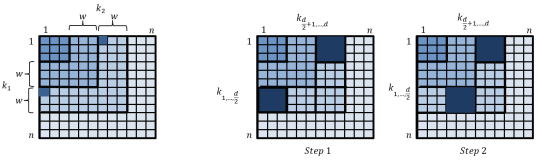

Fig. 2. Left: geometric representation of the key space divided into layers of width $w = 3$. The keys in cells $(1, 7)$ and $(7, 1)$ are the algorithm's seeds for $layer_3^{(3)}$. Right: geometric representation of the key enumeration at $layer_3^3$.

A key observation is that we can run the optimal enumeration algorithm of [22] *within* a layer: we seed the algorithm data structure F by inserting the two "corners" (see Fig. 2), 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:

Proposition 1. *For every $l > 0$ and $w > 0$, applying the optimal key enumeration of [22] on $layer_l^w$, the number of next potential key candidates is bounded by $2w$, i.e., $|F| \leq 2w$.*

Proof. Appears in the extended version of this paper [7].

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

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 [22]. Following the proposition, the number of potential next candidates, F, that our algorithm should store is bounded by $2w$.

4.3 Generalization to a Multi-dimensional Algorithm

For $d > 2$, similarly to [22] 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, 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 $L_{1,2}$ and $L_{3,4}$ on the fly as far as required. For this, we maintain a set of next potential candidates, for each dimension - $F_{1,2}$ and $F_{3,4}$, so that each next subkey candidate we get from $F_{1,2}$ (or $F_{3,4}$) we store at $L_{1,2}$ (or $L_{3,4}$). The length of these generated subkey lists, $L_{1,2}$ and $L_{3,4}$ is $\Omega(n^2)$. For general d, the sizes of the data structures $F_{1,\ldots,d/2}$ and $F_{d/2+1,\ldots,d}$ are bounded by $2w$, however, we still have a bottleneck of $\Omega(n^{d/2})$ because of $L_{1,\ldots,d/2}$ and $L_{d/2+1,\ldots,d}$. Therefore, instead of naively storing the full subkey order of $L_{1,\ldots,d/2}$ and $L_{d/2+1,\ldots,d}$, 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 Fig. 2 (right side). 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 $S_{x,y}^w = \{(k_{1,\ldots,d/2}^{(i)}, k_{d/2+1,\ldots,d}^{(j)}) | (x-1)\cdot w < i \leq x \cdot w \text{ and } (y-1)\cdot w < j \leq y \cdot w\}$. 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 $x = z$.

This in-layer split into squares reduces the space complexity, since instead of storing the full ordered lists of the lower levels, we store only the relevant subkeys candidates for enumerating the current two squares, i.e., $2w$ subkey candidates for each dimension. However, these subkey candidates which are redundant for enumerating the current squares, might be useful later in the enumeration of the next layer. In that case we will need to recompute them.

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 [22] on them (represented in dark gray in step 1 in Fig. 2). 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 of Fig. 2).

Algorithm 1. w-Layer Key Enumeration Algorithm.

Input: Subkey distributions $\{k_i\}_{1\le i\le d}$.
Output: The correct key, if exists, NOT-FOUND otherwise.

```
found = false; initialize(F_{1,...,d});
while (F_{1,...,d} ≠ ∅) do
    candidate = nextCandidate(F_{1,...,d}, {k_i}_{1≤i≤d});
    found = isCorrectKey(candidate);
    if (found) then
        return candidate;
return NOT-FOUND;
```

In the 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 its successor. Without loss of generality, we assume that the successor is in the same row as the current one. Therefore, for all candidates $(k^{(i)}_{1,\ldots,d/2}, k^{(j)}_{d/2+1,\ldots,d})$ 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 redundant and we do not need to store them.

It is simple to see that we maintain at most 2 squares of size $w \times w$ each time, therefore we need to maintain sets of next potential candidates and ordered lists for each square, i.e., $F^1_{1,\ldots,d/2}$, $L^1_{1,\ldots,d/2}$, $F^1_{d/2+1,\ldots,d}$, $L^1_{d/2+1,\ldots,d}$ and $F^2_{1,\ldots,d/2}$, $L^2_{1,\ldots,d/2}$, $F^2_{d/2+1,\ldots,d}$, $L^2_{d/2+1,\ldots,d}$.

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* $m = 1, ..., d$ $(\alpha_m \le 1)$. *Then,*

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

Proof. Appears in the extended version of this paper [7].

Algorithm 2. nextCandidate.

Input: $F_{p,..,r}$ and subkey distributions $\{k_i\}_{p \le i \le r}$.
Output: The next key candidate in $F_{p,..,r}$.

$q \triangleq \lfloor p + r \rfloor / 2$; $x \triangleq \{p, ..., q\}$; $y \triangleq \{q + 1, ..., r\}$;
$(k_x^{(i)}, k_y^{(j)}) \leftarrow$ most likely candidate in $F_{p,...,r}$;
$F_{p,...,r} \leftarrow F_{p,...,r} \setminus \{(k_x^{(i)}, k_y^{(j)})\}$;
$I \triangleq \lceil i \rceil / w$; $J \triangleq \lceil j \rceil / w$; $t \triangleq (I \ge J)$? 1 : 2; // $(k_x^{(i)}, k_y^{(j)})$ is in $S_{I,J}$;
if $S_{I,J}$ *is completely enumerated* **then**
  if $I == J$ **then**
    if $r - p > 1$ **then**
      nextCandidate(F_x^1); $k_x^{(i+1)} \leftarrow L_x^1[(i+1)\%w]$;
      nextCandidate(F_y^2); $k_y^{(j+1)} \leftarrow L_y^2[(j+1)\%w]$;
      $F_x^2 \leftarrow k_x^{(1)}$; $F_y^1 \leftarrow k_y^{(1)}$;
    $F_{p,...,r} \leftarrow \{(k_x^{(1)}, k_y^{(j+1)})\} \cup \{(k_x^{(i+1)}, k_y^{(1)})\}$;
  else
    if *no candidates are in same row/column as Successor*$(S_{I,J})$ **then**
      $F_{p,...,r} \leftarrow F_{p,...,r} \cup \{$most likely candidate in *Successor*$(S_{I,J})\}$;
else
  if $(k_x^{(i+1)}, k_y^{(j)}) \in S_{I,J}$ *and no candidate in row i+1* **then**
    if $r - p > 1$ **then**
      if $k_x^{(i+1)}$ *is not stored at* L_x^t **then**
        nextCandidate(F_x^t);
      $k_x^{(i+1)} \leftarrow L_x^t[(i+1)\%w]$;
    $F_{p,...,r} \leftarrow F_{p,...,r} \cup \{(k_x^{(i+1)}, k_y^{(j)})\}$;
  if $(k_x^{(i)}, k_y^{(j+1)}) \in S_{I,J}$ *and no candidate in column j+1* **then**
    if $r - p > 1$ **then**
      if I==J **then**
        $k_y^{(j+1)} \leftarrow L_y^2[(j+1)\%w]$
      else
        if $k_y^{(j+1)}$ *is not stored at* L_y^t **then**
          nextCandidate(F_y^t);
        $k_y^{(j+1)} \leftarrow L_y^t[(j+1)\%w]$;
    $F_{p,...,r} \leftarrow F_{p,...,r} \cup \{(k_x^{(i)}, k_y^{(j+1)})\}$;
$L[(L.size + 1)\%w] \leftarrow$ most likely candidate in $F_{p,...,r}$;
return $(k_x^{(i)}, k_y^{(j)})$;

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

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

Proof. Appears in the extended version of this paper [7].

It is somewhat counter-intuitive that the bound on the approximation factors does not depend on the size of the layer w, while, as we will see in Sect. 5, the

experimental analysis suggests a much better (yet w-dependent) behavior. We leave for further work to find w-dependent theoretical bounds.

4.5 Parallelization of w-Layer Algorithm

We parallelize our algorithm by parallelizing the OKEA [22] inside each square. OKEA is an inherently serial algorithm, so by parallelizing it we lose the enumeration order's optimality inside the square. However, our bounds on the rank (Theorem 3) and the guessing entropy (Theorem 4) are independent of the internal enumerating order in each layer. Therefore our parallel algorithm retains the same guaranteed performance.

According to Proposition 1, when enumerating a whole layer, the number of next potential candidates, $|F|$, is bounded by $2w$, and within a single $w \times w$ square we have $|F| \leq w$. Hence, enumerating each square can be parallelized between at most w cores, protecting the access to the structure F with concurrency controls. Each core extracts the most likely candidate from F, and let s_1 be one of its two successors. The algorithm inserts s_1 back into F only if there is no candidates in the same row/column as s_1, and if all the candidates in the same row/column, before s_1, were already enumerated. For this, we need to replace the simple "bit vector" implementation of [22] by a "greatest index vector". This vector stores for any row/column the greatest enumerated index in that row/column.

4.6 Space Complexity Analysis

The algorithm needs to store the candidates of the 2 top-level dimensions, for each dimension. However, it doesn't need to store the whole candidate list, but only two lists (L) of size w for each dimension. For this, it needs to store 2 sets of potential candidates (F) for each dimension, each one of these sets is bounded by $2w$. Moreover, it needs to store 2 data structures ("bit vectors") for each F to indicate which rows and columns currently have a member in it. All together, we get the following recurrence relation for the space complexity:

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

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 Performance Analysis

We evaluated the performance of our w-layer key enumeration algorithm through an extensive simulation study. We implemented the optimal algorithm [22] and our algorithm in Java, and ran both algorithms on a 3.07 GHz PC with 24 GB RAM running Microsoft windows 7, 64 bit. 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 $d = 8$ subkeys and $n = 2^{12}$ candidates per subkey for a total enumeration space of $O(n^d) = 2^{96}$. We chose $d = 8$ and $n = 2^{12}$ since for a key whose rank is 'deep', the optimal-order algorithm takes space of $\Omega(n^{d/2}) = \Omega(2^{48})$ which exceeds the available memory. We generated the synthtic SCA distributions according to Pareto distributions, with $\alpha = 0.575$ and $\beta = 0.738$. The choice of the Pareto distribution and these specific parameters is based on empirical evidence we discovered, see the extended version of this paper [7]. For the simulations, we chose two different values of w to limit our space complexity $O(d^2w + dn)$. The first one is $w = n$ which gives a linear space complexity of $O(d^2n + dn)$ and the second one is $w = 2^{25}$ which gives an $O(2^{31})$ space complexity which is about 1 Gb.

We also evaluated the algorithm's performance for $d = 16$ subkeys and $n = 2^6$, again for a total enumeration space of 2^{96}. The probability distributions were Pareto distributions with $\alpha = 0.3$ and $\beta = 1.1197$, see the extended version of this paper [7]. We analyzed our w-layer algorithm for two different values of w: $w = n = 2^6$ and $w = 2^{25}$. 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.

Because of time consumption, we decided to stop each w-layer run after 2 h - if it didn't find the given key by then. We marked the timed-out runs.

5.1 Runtime Analysis

Figure 3 illustrates the time (in minutes) of the 3 algorithms: OKEA (optimal-order) (green triangles), w-layer with $w = n = 2^{12}$ (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. The w-layer key enumeration, in contrast, keeps running.

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 ($w = n$) in Fig. 3 are misleadingly low, since as Fig. 4 shows, a large fraction of runs timed-out at the 2 h 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 finds the correct key.

For the w-layer algorithm with $w = 2^{25}$, however, we see excellent results. For small ranks it takes exactly the same time consumption as OKEA, (hidden by the green triangles in Fig. 3), and for high ranks, its bounded space complexity enables it to enumerate in reasonable time.

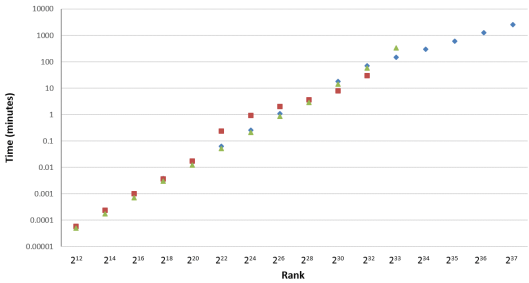

Fig. 3. Median run time, in minutes, of OKEA (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks. (Color figure online)

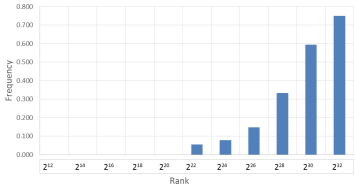

Fig. 4. Frequency of the keys whose time consumption applying the w-layer key enumeration with $w = n$ is higher than 2 h.

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 to 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 h time out.

We can see that bigger values of w lead to more candidates in each w-layer which leads to less recomputing and therefore a lower running time.

5.2 Space Utilization

Figure 5 illustrates the space (in bytes) used by the 3 algorithms' data structures for different ranks. As we can see again, OKEA stops at 2^{33} because of memory shortage, while the w-layer algorithm keeps running. For the w-layer algorithm with $w = n$ we can clearly see the bounded space consumption leveling at around 1 MB. For the w-layer algorithm with $w = 2^{25}$ we see that its space consumption levels around 4 GB and remains steady, allowing the algorithm to enumerate further into the key space, limited only by the time the cryptanalyst is willing to spend.

5.3 The Difference in Ranks

Figure 6 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

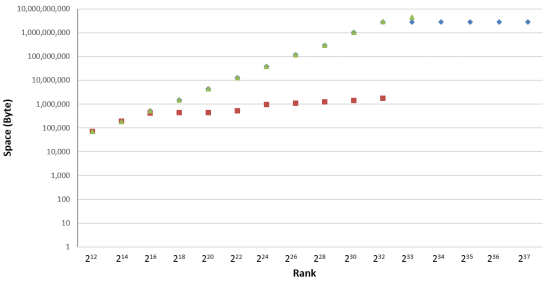

Fig. 5. Median space, counting the data structure elements, of OKEA (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks. (Color figure online)

the somewhat pessimistic bounds of Theorem 4, the figure shows that with $w = n$ the ratio between the optimal rank and $rank^w$ is approximately 2.32 (again, for those runs that complete faster than 2 h running time). 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.

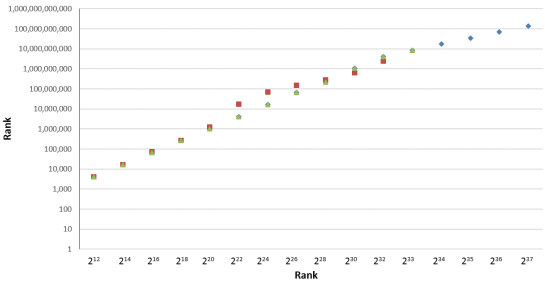

Fig. 6. Median rank of OKEA (green triangles), w-layer key enumeration with $w = 2^{25}$ (blue diamonds) and w-layer key enumeration with $w = n$ (red squares) on different ranks. (Color figure online)

5.4 Influence of w on Space Complexity and Enumeration Accuracy

The trade-off between the space complexity and the accuracy of the enumeration order is summed up in Table 1. As we can see, for $w = 2^{12}$ our enumeration uses space of 1 MB. We see the maximum rank for which 80% of the simulations take less than 2 h is 2^{26}, and up to this rank the rank accuracy is at most 2.32 times the optimal rank. For $w = 2^{25}$ our enumeration uses more space (4 GB), but the maximum rank for which 80% of the simulations take less than 2 h is 2^{33}, and accuracy is at most 1.007 times the optimal rank. As a consequence, we recommend to increase w as much as possible without exceeding the available

Table 1. Influence of w on space complexity and enumeration accuracy

w	Space	Max rank of 80% in 2 h	Accuracy
$w = 2^{12}$	1 MB	2^{26}	$\leq 2.32 \cdot$ OPT
$w = 2^{25}$	4 GB	2^{33}	$\leq 1.007 \cdot$ OPT

memory. This bounds the space complexity, and therefore enables to enumerate more keys, with better accuracy.

6 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 parallelizable 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 [22], 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.

Along the way, we also provided bounds on the full key guessing entropy in terms of the guessing entropies of the individual subkeys.

Finally, an open-source Java implementation for both our w-layer key enumeration and the order-optimal enumeration [22] are available via the authors' home pages.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side—channel(s). In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Berlin (2003). doi:10.1007/3-540-36400-5_4
2. Anonymous: Anonymous (2015)
3. Bernstein, D.J., Lange, T., van Vredendaal, C.: Tighter, faster, simpler side-channel security evaluations beyond computing power. Cryptology ePrint Archive, Report 2015/221 (2015). http://eprint.iacr.org/
4. Bogdanov, A., Kizhvatov, I., Manzoor, K., Tischhauser, E., Witteman, M.: Fast and memory-efficient key recovery in side-channel attacks. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 310–327. Springer, Cham (2016). doi:10.1007/978-3-319-31301-6_19
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Berlin (2004). doi:10.1007/978-3-540-28632-5_2

6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Berlin (2003). doi:10.1007/3-540-36400-5_3
7. David, L., Wool, A.: A bounded-space near-optimal key enumeration algorithm for multi-dimensional side-channel attacks. Cryptology ePrint Archive, Report 2015/1236 (2015). http://eprint.iacr.org/2015/1236
8. Dichtl, M.: A new method of black box power analysis and a fast algorithm for optimal key search. J. Crypt. Eng. **1**(4), 255–264 (2011)
9. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Berlin (2001). doi:10.1007/3-540-44709-1_21
10. Glowacz, C., Grosso, V., Poussier, R., Schüth, J., Standaert, F.-X.: Simpler and more efficient rank estimation for side-channel security assessment. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 117–129. Springer, Berlin (2015). doi:10.1007/978-3-662-48116-5_6
11. Junod, P., Vaudenay, S.: Optimal key ranking procedures in a statistical cryptanalysis. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 235–246. Springer, Berlin (2003). doi:10.1007/978-3-540-39887-5_18
12. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Berlin (1999). doi:10.1007/3-540-48405-1_25
13. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Berlin (1996). doi:10.1007/3-540-68697-5_9
14. Martin, D.P., O'Connell, J.F., Oswald, E., Stam, M.: Counting keys in parallel after a side channel attack. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 313–337. Springer, Berlin (2015). doi:10.1007/978-3-662-48800-3_13
15. Meier, W., Staffelbach, O.: Analysis of pseudo random sequences generated by cellular automata. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 186–199. Springer, Berlin (1991). doi:10.1007/3-540-46416-6_17
16. Oren, Y., Weisse, O., Wool, A.: A new framework for constraint-based probabilistic template side channel attacks. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 17–34. Springer, Berlin (2014). doi:10.1007/978-3-662-44709-3_2
17. Pan, J., Woudenberg, J.G.J., Hartog, J.I., Witteman, M.F.: Improving DPA by peak distribution analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 241–261. Springer, Berlin (2011). doi:10.1007/978-3-642-19574-7_17
18. Poussier, R., Grosso, V., Standaert, F.-X.: Comparing approaches to rank estimation for side-channel security evaluations. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 125–142. Springer, Cham (2016). doi:10.1007/978-3-319-31271-2_8
19. Poussier, R., Standaert, F.-X., Grosso, V.: Simple key enumeration (and rank estimation) using histograms: an integrated approach. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 61–81. Springer, Berlin (2016). doi:10.1007/978-3-662-53140-2_4
20. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Berlin (2001). doi:10.1007/3-540-45418-7_17

21. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Berlin (2009). doi:10.1007/978-3-642-01001-9_26
22. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.-X.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Berlin (2013). doi:10.1007/978-3-642-35999-6_25
23. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Security evaluations beyond computing power. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 126–141. Springer, Berlin (2013). doi:10.1007/978-3-642-38348-9_8
24. Ye, X., Eisenbarth, T., Martin, W.: Bounded, yet sufficient? how to determine whether limited side channel information enables key recovery. In: Joye, M., Moradi, A. (eds.) CARDIS 2014. LNCS, vol. 8968, pp. 215–232. Springer, Cham (2015). doi:10.1007/978-3-319-16763-3_13