Scalable Key Rank Estimation (and Key Enumeration) Algorithm for Large Keys

Vincent Grosso

Scalable Key Rank Estimation (and Key Enumeration) Algorithm for Large Keys

Vincent Grosso*

Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, Saint-Etienne - France.

Abstract. Evaluation of security margins after a side-channel attack is an important step of side-channel resistance evaluation. The security margin indicates the brute force effort needed to recover the key given the leakages. In the recent years, several solutions for key rank estimation algorithms have been proposed. All these solutions give an interesting trade-off between the tightness of the result and the time complexity for symmetric key. Unfortunately, none of them has a linear complexity in the number of subkeys, hence these solutions are slow for large (asymmetric) keys. In this paper, we present a solution to obtain a key rank estimation algorithm with a reasonable trade-off between the efficiency and the tightness that is suitable for large keys. Moreover, by applying backtracking we obtain a parallel key enumeration algorithm.

1 Introduction

Side-channel attacks are powerful attacks against cryptographic implementations. To perform a side-channel attack, an attacker needs to be able to measure some physical properties (e.g. power consumption, electromagnetic radiation) of the device while it computes some key dependent operations. With this additional information, some attacks can be performed against cryptographic implementations. Hence, cryptographic algorithms required secure implementations.

To evaluate the security margin, evaluation labs generally launch some popular attacks to evaluate if an adversary can break an implementation by performing, for example, a key recovery attack. This approach is adapted since the leakage of an implementation dependents on the device. Thus, the security obtained by an implementation is highly dependent on the underlying device.

Most of state of the art side-channel attacks follow a divide-and-conquer strategy, where the master key is split into several pieces, called subkeys. The attacker/evaluator mounts an independent attack for each of these subkeys. He then needs to combine the different results of the attacks. A security evaluation only based on a success or failure of a key recovery attack is limited by the computational power of the evaluator. To get rid of this limitation a solution is to compute the rank of the key instead of performing a key recovery attack. The

* Part of this work was done while the author was at Radboud University Nijmegen, Digital Security Group, The Netherlands.

rank corresponds to the number of keys needed to be tested before recovering the actual key. Recently, several papers studied how to evaluate the security by evaluating the computational power required after a side-channel attack [1,8,11,15]. These papers compute an estimation of the rank of the key after a side-channel attack, without being limited by the evaluator computational power. All these papers focus on symmetric key size. In [8] the authors managed to evaluate ranks for 1024-bit keys, but for larger keys, this solution could have some limitations.

Our contributions. We study the cost of the solution of Glowacz et al. for large keys. Next, we present a variation of this key rank estimation algorithm. This variation allows us to obtain a linear complexity of the algorithm in the number of subkeys. We then derive some tighter bound for our construction. These tight bounds allow us to have an efficient and tight solution for key rank estimation for large keys (size greater than 1024 bits). Finally, by applying a similar idea as Poussier et al. [13], we propose a new key enumeration algorithm.

2 Background

2.1 Side-channel attacks and notations

For the rank estimation/key enumeration problems, the details on the divide-and-conquer attack are not necessary. We just need to specify the output of the attack. Let us assume that the attacker targets a $\boldsymbol{\eta}$**-bit master key**. An adversary using a divide-and-conquer strategy will split this key into $\boldsymbol{\nu}$ **subkeys** of (for simplicity equally sized) $\boldsymbol{\kappa}$ **bits of subkey**. For each subkey k_i the attacker will obtain a list of probability for each possible value of the key $\mathcal{L}_i = \{\Pr[k_i = 0|\text{SCI}], \ldots, \Pr[k_i = 2^\kappa - 1|\text{SCI}]\}$, where SCI stands for the side-channel information the adversary obtained. Divide-and-conquer strategy is useful as $\nu \times 2^\kappa$ is smaller than 2^η. Note that if the adversary scores instead of probability he could either use a Bayesian extension [14] or use direct results [4].

2.2 Key enumeration algorithms

From the result of an attack, either all the correct subkeys have the highest probability of the list of the candidate subkeys or the attacker need to test the most likely keys. Some solution exists to recombine this information in a smart way [2,6,10,11,13,14]. All these algorithms have been tested in a symmetric key setting and provide efficient solution.

The algorithms proposed in [2,11,13] can be separate in two phases: a construction phase (that is similar to key rank estimation) and a backtracking part that enumerates the keys. For symmetric keys setting, the first part (construction) is negligible in comparison to the second (backtracking).

2.3 Rank estimation algorithms

A rank estimation algorithm is a tool that allows an evaluator to estimate the brute force an attacker need to perform a successful attack, i.e. how many keys the attacker needs to test in the recombination phase before she recovers the actual key (the key is known by the evaluator). As we want to evaluate security against a smart adversary we should assume that she can enumerate the keys from the most probable one to the least probable one (but still in its computational power limits).

Definition 1 (Rank of the key). *The rank of the key k after a side-channel attack is defined as the number of keys that have a higher probability than k.*

$$rank(k) = \#\{k^* | \Pr[k^*|\text{SCI}] \geq \Pr[k|\text{SCI}]\}.$$

Where # stands for the cardinality of the set.

Definition 2 (Tightness). *The tightness of an estimation is the logarithm of the ratio between the upper and lower bound* $\log_2 \left(\frac{rank\_upper\_bound}{rank\_lower\_bound} \right)$.

In the rest of this paper, the probability of a key is equal to the product of the probabilities of its subkeys. Hence, we suppose that the subkeys probabilities are independent, and so the different attacks.

The main advantage of using a key rank estimation algorithm is that an evaluator does not need to perform the brute force search to estimate the costs of such a search. In the past few years, several solutions have been proposed to solve this problem [1,8,11,15]. Efficient rank estimation algorithms [1,8,11] share the same step that introduces error: they map the probabilities to integers (see [12] for a discussion on the errors introduced by algorithms that calculate security margins). Using this simplification they can estimate the rank of the key quite efficiently, with bounded error due to some truncation that appears during the conversion from real (float) to integer. Hence, these algorithms cannot compute the rank, but an upper bound (*rank_upper_bound*) and a lower bound (*rank_lower_bound*) of the rank. These rank estimation algorithms are based on samples, i.e.they use result of an attack and calculate bounds on the rank. To obtain some indication of the security level of the device several experiments attacks are launched and results could be displayed in a security graph as proposed in [15].

Some other solutions exist to evaluate the security of a device that can be faster and adaptable for large keys [7,16]. These solutions are based on metrics, i.e. do not use directly result of an attack, but use results of several attacks to compute a metric e.g. the success rate. However, solutions based on metric could misestimate the actual computational power of an attack, as pointed out in [12].

2.4 The histogram solution

Since our solution is based on the Glowacz et al. solution [8], we give some more highlight on this solution. In the rest of the paper, we refer to this solution as FSE'15. The different steps of this algorithm can be summarized as follow:

1. *from multiplicative relation to additive relation*: since the subkeys are independent we have $\Pr[k_1, k_2|SCI] = \Pr[k_1|SCI] \times \Pr[k_2|SCI]$. By using logarithm, we have $\log(\Pr[k_1, k_2|SCI]) = \log(\Pr[k_1|SCI]) + \log(\Pr[k_2|SCI])$.
2. *from reals to integers*: in the FSE'15 solution, this step is done by casting the results of the side-channel attacks into histograms. For each subkey a histogram is built, the histograms should have the same bin size. The bin height corresponds to the number of candidate subkeys that have a log probability included between the limits of the bin.
3. *convolution of histograms*: the convolution of histograms gives us the distribution of the combination of the probabilities of different combination of subkeys. Remark the height of i-th bin of H_3 that is the result of the convolution of H_1 and H_2 is $H_3(i) = \sum_j H_1(j) \times H_2(i-j)$. This is the number of couples of subkey candidates that have the sum of the estimated sum of log probabilities that correspond to the center of the bin i.
4. *calculate bound*: This is done by summing the bins that represent a higher log probability than the bin of the key's log probability ($\pm$ the error bounds). Hence, having tight error bounds allow obtaining tighter results.

In listing 1 we give a simplified version of the code of the two last steps.

Listing 1. Matlab implementation of FSE'15 solution.

```
function [mini,maxi] = rank(hi,b)
% Inputs:
%hi: list of histogram score for each subkey (hi(subkey,:))
%b: bin index of the log probability of the actual key
%Outputs     Mini the minimum rank of the key
%            Maxi the maximum rank of the key
[dim,~]=size(hi);
H=conv(hi(1,:),hi(2,:));
for i=3:dim
    H=conv(H,hi(i,:));
end
mini=sum(H(b+(dim/2)+1:length(H)));
maxi=sum(H(b-dim/2:length(H)));
end
```

Since the histograms put every log probabilities in the bin center some error could appear. In [8] the authors show that the maximum distance in numbers of bin between a bin of a sum of log probabilities and the bin where the FSE algorithm could put it is $\frac{\nu}{2}$. That is why the minimum and maximum are shifted by such a value.

Example 1. Let us assume we have two subkeys k_1, k_2 of 3 bits. With the probabilities given in Table 1. As our histograms will use the logarithm of the probabilities (to have an additive relation), we also provide the logarithm values and also the key candidates' bin.

Table 1. Probabilities of subkeys candidates and their logarithm and bin values.

	k_1			k_2		
Candidate	Pr	log	bin	Pr	log	bin
0	0.6643	-0.5901	1	0.0012	-9.7027	3
1	0.2588	-1.9501	1	0.0011	-9.8283	3
2	0.0313	-4.9977	2	0.3588	-1.4787	1
3	0.0412	-4.6012	2	0.0713	-3.8100	1
4	0.0001	-13.2877	4	0.5643	-0.8255	1
5	0.0020	-8.9658	3	0.0012	-9.7027	3
6	0.0013	-9.5873	3	0.00005	-14.2877	4
7	0.0010	-9.9658	3	0.00205	-8.9302	3

We construct the histograms as follows. The bin 1 corresponds to the number of keys with logarithm probabilities between -16 and -12, the bin 2 corresponds to the number of keys with logarithm probabilities between -12 and -8, the bin 3 corresponds to the number of keys with logarithm probabilities between -8 and -4 and the bin 4 corresponds to the number of key with logarithm probabilities between -4 and 0. The histograms are displayed in Figure 1. H_1 is the histogram

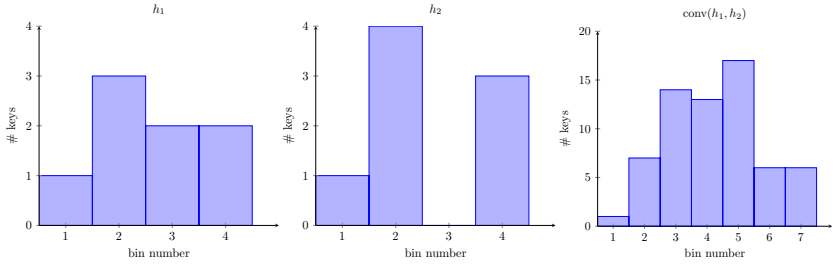

Fig. 1. The histograms for the two subkeys and the convolution result.

for the subkey candidates of k_1. The sum of the bins gives us 8 that is the number of subkey candidates. H_2 is the histogram for the subkey candidates of k_2.

Then by performing the convolution we have the distribution of all possible couple for the subkeys (k_1, k_2). In the histogram of Figure 1, the bin 1 should correspond to the number of couples of candidate keys with logarithm probabilities between -30 and -26, since we only look at the center of the bin some error could appear here.

Part 3 (for loop line 9 in listing 1) of such an algorithm is the most expensive part . We need to perform $nb\_subkeys - 1$ convolution, each convolution having a cost in $nlog(n)$ when FFT is used. Remark this n is the size of the outputted histogram (and thus on the number of convolutions already performed), that

means the cost of convolution became more and more expensive as the size of the histogram H grows. It comes out that the cost of the rank estimation of Glowacz et al. grows not linearly with the number of subkeys. This observation is validated by experiments in Section 4.

During the computation, we need to use large numbers (a bin can contain a number between 0 and 2^η). Hence, to avoid precision error due to large number we need to use large integer library and/or the Chinese remainder theorem as proposed in Appendix B of [8].

Another limitation for large keys is the size of the histogram that will grow linearly in the number of subkeys. After the convolution i-th the size of the histogram H is of size $(i-1) \times dim$, the value stored in that table could go up to 2^η. This could be expensive for large key and high precision. The FSE'15 solution needs to store the last histogram.

Example 2. That means for histograms with 2^{16} bins and for a key of 256 subkeys, we need to store a table of $\simeq 2^{24}$ values. These values are integers of at most 2048 bits (if subkeys are bytes). That means around 4GB.

If the size of the key doubles the memory required double. Remark for the enumeration all intermediate histograms need to be stored to apply the backtracking solution this could require some large amount of memory.

3 Scalable rank estimation algorithm

The main idea of our solution is to keep histogram with a constant number of bins. This is achieved by batching two by two the bins of the convolution's result histograms (line 15 in listing 2).

Listing 2. Matlab implementation of our solution.

```
function [mini,maxi] = rank(hi,b)
% Inputs/output same as Listing 1
[dim,~]=size(hi);
H2=cell(log2(dim),dim/2);
for i=2:2:dim
    H=conv(hi(i-1,:),hi(i,:));
    H2{1,i/2}=[H(2:2:length(H)),0]+H(1:2:length(H));
end
dim=dim/2;
j=1;
while dim>1
    j=j+1;
    for i=2:2:dim
        H=conv(H2{j-1,i-1},H2{j-1,i});
        H2{j,i/2}=[H(2:2:length(H)),0]+H(1:2:length(H));
        end
        dim=dim/2;
end
mini=sum(H2{j,1}(b+error(dim)+1:length(H2{j,1})));
```

```
maxi=sum(H2{j,1}(b+error(dim):length(H2{j,1})));
end
```

Where the error function is a function that gives the approximation error due to our casting and batching. This function and the values outputted are discussed in Subsection 3.3.

As for the FSE'15 solution we perform convolution on histograms and obtain the histogram H, but after this step we batch bins in pairs and obtain the histogram H_2. The i-th bin of H_2 is equal to the sum of the $2i$-th and the $2i+1$-th bins of H, $H_2(i) = H(2i) + H(2i+1)$. Doing so H_2 has the same number of bins as the initial histogram. But then the bin size of the histogram after the batching is twice as large as the bin size of the loop input histograms.

For rank estimation, we need to perform convolution between histogram with equally sized bins. By performing the batching we increase the width of the bins. To solve the problem we use a recursive approach, we do convolutions of histograms two by two, batch and start a new level of convolutions. Hence we perform convolution in a tree like structure, see the right part of Figure 2. The tree like structure can be used without batching to have similar result as FSE'15.

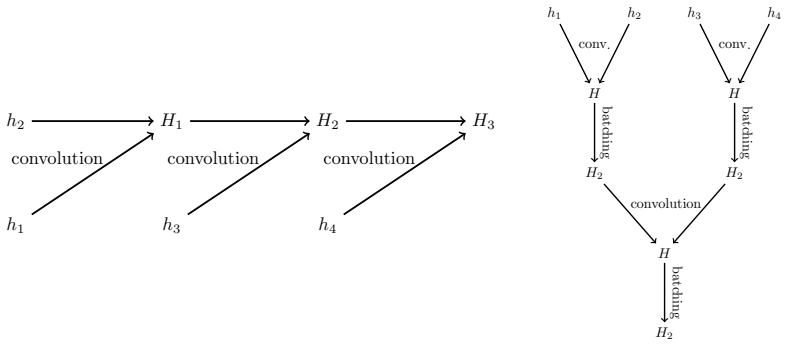

Fig. 2. Representation of the FSE'15 solution (left) and ours (right).

Example 3. In our example 1, the batching step outputs the histogram in figure 3. The batching step merge bin 2 by 2. That means the first bin in the new histogram corresponds to the sum of the bins 1 and 2 from the result of the convolution histogram of figure 1.

3.1 On the time complexity

Our algorithm performs the same number of convolutions as FSE'15. But in our solution, the size (i.e. the number of bins) of the histogram stays the same, the bin size increase.

While for the FSE'15 solution the size of H_i histograms grows, $size(H_i) = ((i+1) \times nb\_bin\_init) - i$, the size of the H_2 histograms in our solution stay the same as the initial histograms, i.e. $size(H_2) = nb\_bin\_init$. That means that

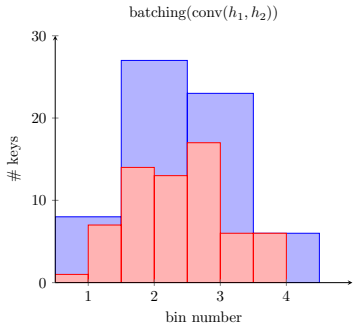

Fig. 3. The batched result.

convolution in level 1 of the tree (right part of figure 2) should require similar computation as the convolution in the last level. Thus, we expect for our solution to have a time that grows linearly with the number of subkeys. This is verified by experiments in Subsection 4.1.

3.2 On memory complexity

As we can see on the figure 2 our method is a tree exploration. That means we can explore it in breadth first or in depth first search.

In the case of a breadth first search, the most expensive step we have to store is the batched histograms after the first step of convolutions, in that case, we need to store $\frac{\nu}{2}$ tables of *nb_bin_init* values. If the size of the key doubles the memory required double.

Example 4. For the same values as example 2, 2^{16} bins and 256 subkeys, we need to store 2^{23} values. That is around 2 GB.

In the case of depth first search, we need to store at most one batched histogram per level ($\log_2 \nu$). If the size of the key doubles the memory required increase by one histogram.

Example 5. For the same values as example 2, we need to store 2^{19} values. That is around 128 MB.

For simplicity we describe in listing 2 the breadth first search. Both breadth and deep first technique have similar time, the choice of one over the other is then based on memory available.

3.3 Bounded error

The tight bounds we obtain for our method lead to efficient tight results. The error is introduced when we cast real numbers into integers as for FSE'15. Our solution also introduces error when the batching step is performed.

For the rounding error that appears when we transform real numbers into integers. For every log probability of a subkey candidate $k = i$ and for histogram of bin width 2ϵ there exist a bin b_i of center c_i such that $c_i - \epsilon \leq \log(\Pr[k = i]) \leq c_i + \epsilon$.

If we look at the combined candidate $(k_1 = i, k_2 = j)$ we know that for the initial histogram we have:

$$c_i - \epsilon \leq \log(\Pr[k_1 = i]) \leq c_i + \epsilon$$
$$c_j - \epsilon \leq \log(\Pr[k_2 = j]) \leq c_j + \epsilon.$$

By summing the inequalities we obtain:

$$c_i + c_j - 2\epsilon \leq \log(\Pr[k_1 = i]) + \log(\Pr[k_2 = j]) \leq c_i + c_j + 2\epsilon.$$

The convolution will consider that the couple $(k_1 = i, k_2 = j)$ has log probability $c_i + c_j$. Hence the distance between the real log probability and the log probability considered by the convolution is 2ϵ. If we add ν of such inequalities the distance between the real log probability and its bin is bounded by $\nu\epsilon$. That is the bound of the FSE'15 method.

In our case we have also to consider the batching step. Remark when we batch the bins of width w center c_i, c_{i+1} (resp. c_{i-1}, c_i), the new center is $\frac{c_i + c_{i+1}}{2} = c_i + \frac{w}{2}$ (resp. $\frac{c_{i-1} + c_i}{2} = c_i - + \frac{w}{2}$). That means we have the inequality:

$$c_i - \frac{w}{2} \leq batch(c_i) \leq c_i - \frac{w}{2}.$$

Putting the two errors for each level in our tree we double the error of the histograms inputs and add an error of half bin width of histogram inputs. For the first level, we will have:

$$\begin{aligned} batch(c_i + c_j) - 3\epsilon \leq c_i + c_j - 2\epsilon \leq \log(\Pr[k_1 = i]) + \log(\Pr[k_2 = j]) \\ \leq c_i + c_j + 2\epsilon \leq batch(c_i + c_j) + 3\epsilon. \end{aligned}$$

By iterating the error propagation we obtain error for our method. Remark the error can be, more efficiently, computed by the following formula if the input histograms at the first level have bin width 2ϵ:

$$error = \nu\epsilon + \lceil \log_2(\nu) \rceil \frac{\nu}{2}\epsilon.$$

Remark that the final histogram has bin width of $2^{\nu+1}\epsilon$

In FSE'15 the error was given as a number of bin, in our case doing so we will have overestimated margins. Calculate the lower and upper bins from the log probability of the key $\pm$ error give tighter margins.

As for FSE'15 if we double the number of bins we reduce by two the error.

3.4 Non power of 2 cases

If the number of subkeys is not a power of two our first convolution step (line 5 in listing 2) should be adapted.

During the first step of convolutions, we perform a reduced number of convolutions such that at the end of this step the number of histograms is a power of 2. To keep the histograms with the same bin size we need to perform batching on all histograms, even the ones that do not go to the first convolution loop. We refer to longer version of this paper for more details.[1]

4 Experiments

We compare the efficiency of different approaches of key ranking based on histograms (i.e. FSE'15 and our method) in terms of time efficiency and precision.

In all our experiment we consider simulations. We target the memory loading of the key (or subkeys). The memory load target seems to fit the assumption of independence of subkeys for large keys. Note that such attacks have been used for attacks against AVR XMEGA [5]. These attacks do not use the structure of the cipher so can be adapted to asymmetric key implementations at a cost of more computation (linear in the key size). We assume that the attacker was able to perfectly recovered the leakage function.

For our experiments, we have a set of parameters that we modify that we detailed hereafter.

- *The number of subkeys.* The number of subkeys is the principal parameters we want to compare.
- *The precision.* The precision is an important point of comparison for rank estimation algorithms. In our case, we compare histogram based solution the precision is the number of bins.
- *The leakage function.* As we target the memory load of a subkey we can observe only one output of the leakage function $\mathcal{L}$. The only observation we get is $\mathcal{L}(k) + \mathcal{N}$, where k is the subkey and $\mathcal{N}$ is some noise. If we perform several measurements for the same subkey we will observe the same deterministic part of the leakage. Hence, if $\mathcal{L}(k_1) = \mathcal{L}(k_2)$, we will obtain the same probability for k_1 and k_2. Such a property will impact the tightness result of any key rank algorithm that targets such values.
- *The noise.* We consider white Gaussian noise with different variance noise.
- *The size of the subkeys.* For our experiments, we target 8-bit subkeys.

4.1 Same precision

We compare in term of efficiency our method versus the FSE method. In this experiment we look at the tightness and time of our method with 2^{16} bins per histogram at the beginning, FSE'15 with the same amount of bins and FSE'15

[1] `https://eprint.iacr.org/2018/175`

with less bins $\left(\frac{2^{16}}{\nu}\right)$ such that the final histogram have a similar amount of bins as our method. The choice of 2^{16} bins per histogram at the beginning is motivated by the fact this gives quite tight bound in an efficient manner for FSE'15.

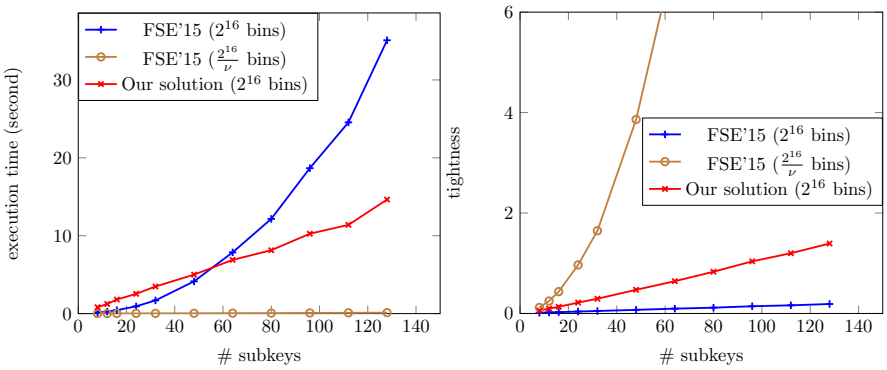

Fig. 4. Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for different sizes of keys (16 bits of precision) and an SNR of 8.

On the graph, we can see that the FSE'15 solution with a constant number of bins (2^{16}) have an execution time that grows faster than linearly, but it is the solution that offers the tightest bound. However, if we use FSE'15 with the same number of bins for the final histogram the solution is quite efficient but the tightness explodes for large keys. Our method seems to have a linear time complexity and a linear increase of the tightness in the size of the key.

4.2 Similar tightness

We compare our method to the FSE'15 method to obtain similar tightness. We look at two levels of tightness 1 bit and 0.3 bit. To obtain similar tightness when the size of the key increase we need to increase the number of bins of the initial histograms. The results are plotted in Figure 5.

The first observation we can make is that the tighter we want the rank estimation, the smallest is the ratio between the time gap between our method and FSE'15. Secondly, since we need to increase the number of bins of the initial histograms the time complexity grows faster than linearly even for our method. However, for a large number of subkeys our solution more efficient than the FSE'15 solution.

4.3 NTL implementation

Matlab implementation of the solution has some limitations mainly due to the fact that large integers are stored in doubles. That means that bins cannot be

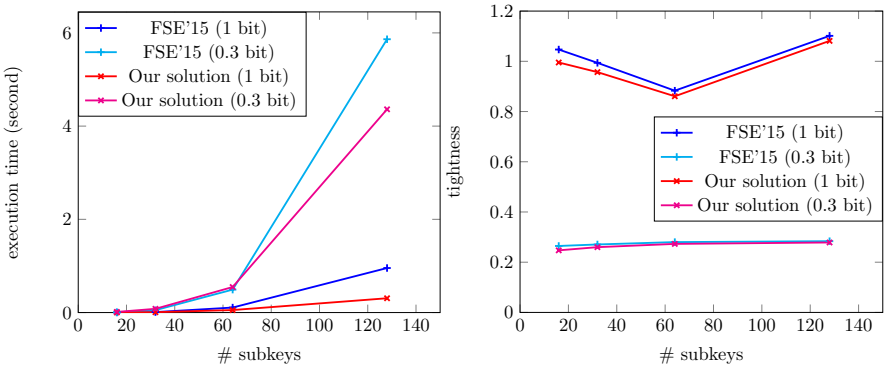

Fig. 5. Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for similar tightness.

higher than 2^{1024}. Thus for large keys (>1024-bit), the implementation could lead to an incorrect result. To solve this problem Glowacz et al. [8] suggest to use Chinese remainder theorem.

To override these issues we implement our solution using a big integer library: the NTL library. We look at histograms starting with 2^{12} and perform the convolution of : 16, 32, 64, 128, 256, 512 and 1024 histograms. For the classical FSE'15 method for the 128 convolutions and an initial number of bins 2^{12} we get an error message saying that histograms where too large (the number of bins) to perform the convolution. Our C implementation allows to obtain rank for very large keys (up to 1024 subkeys in less than 15 seconds).

4.4 Comparison with CHES 2017

At CHES 2017 Choudary and Popescu present an "impressively fast, scalable and tight security evaluation tools" [3]. Note that their tool does not calculate the rank of the key but the expected value of the rank. As pointed out in [9] it is not clear how to evaluate the power computation required to recover the key from the expected value of the rank. This is mainly due to the distribution of the rank that is not easy to model. However, we want to compare our method, the FSE'15 method and the CHES 2017 method in terms of efficiency/tightness. As the CHES 2017 do not offer parameters to tighten the bounds we play with the number of bins for FSE and our method to have similar tightness. The results are plotted in Figure 6.

We can see that indeed the CHES'17 solution is quite efficient. In the same time, for such a tightness all solutions run in less than 100ms for 128 subkeys. For such bounds, it seems that the rank computation's time is not the bottleneck of an evaluation.

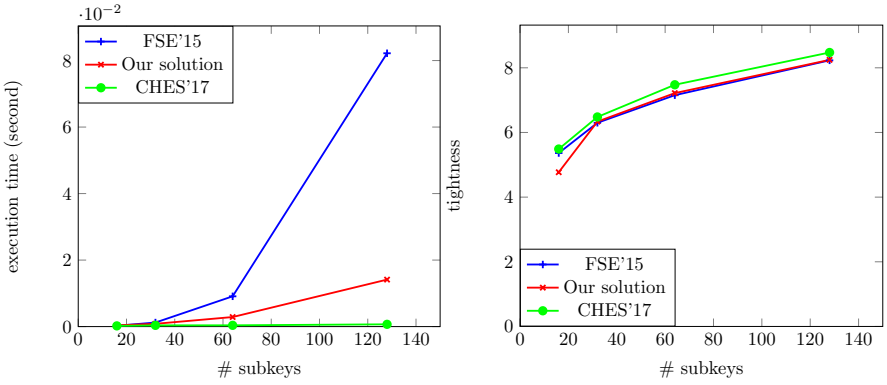

Fig. 6. Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for similar tightness as CHES'17.

5 Key enumeration

We can apply similar idea as the backtracking used in [13]. Our technique speed up the construction phase of a solution like [13]. In general, this step is negligible for key enumeration algorithm. We refer to longer version of this paper for more details.[2] However, our enumeration algorithm has an advantage when memory needed to store histograms is too large.

6 Conclusion

We present a trick to reduce the cost of rank estimation for a large number of subkeys based on the rank estimation of [8]. It can be applied to evaluate security against side-channel of cryptographic implementation that uses large keys. Our solution has the advantage to have a linear complexity in the number of subkeys. Our method allows to estimate efficiently rank of the key thanks to the tight bounds we manage to evaluate. Finally, our algorithm could be used as a construction phase for an enumeration algorithm. This algorithm could be useful when the number of subkeys if large and thus classical enumeration algorithm required a large amount of memory. Finally, our error bound estimation could be applied to other cases, in particular we can look at not equally sized histograms.

Acknowledgments. I thank the anonymous reviewers and Mathieu Carbone, Romain Poussier and François-Xavier Standaert for the improvements pointed out.

[2] `https://eprint.iacr.org/2018/175`

References

1. Bernstein, D.J., Lange, T., van Vredendaal, C.: Tighter, faster, simpler side-channel security evaluations beyond computing power. IACR Cryptology ePrint Archive 2015, 221 (2015), http://eprint.iacr.org/2015/221
2. 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.) Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9566, pp. 310–327. Springer (2015), https://doi.org/10.1007/978-3-319-31301-6_19
3. Choudary, M.O., Popescu, P.G.: Back to massey: Impressively fast, scalable and tight security evaluation tools. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 367–386. Springer (2017), https://doi.org/10.1007/978-3-319-66787-4_18
4. Choudary, M.O., Poussier, R., Standaert, F.: Score-based vs. probability-based enumeration - A cautionary note. In: Dunkelman, O., Sanadhya, S.K. (eds.) Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10095, pp. 137–152 (2016), https://doi.org/10.1007/978-3-319-49890-4_8
5. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8419, pp. 253–270. Springer (2013), https://doi.org/10.1007/978-3-319-08302-5_17
6. David, L., Wool, A.: A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In: Handschuh, H. (ed.) Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10159, pp. 311–327. Springer (2017), https://doi.org/10.1007/978-3-319-52153-4_18
7. Duc, A., Faust, S., Standaert, F.: Making masking security proofs concrete - or how to evaluate the security of any leaking device. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 401–429. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_16
8. Glowacz, C., Grosso, V., Poussier, R., Schüth, J., Standaert, F.: Simpler and more efficient rank estimation for side-channel security assessment. In: Leander, G. (ed.) Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9054, pp. 117–129. Springer (2015), https://doi.org/10.1007/978-3-662-48116-5_6
9. Martin, D.P., Mather, L., Oswald, E., Stam, M.: Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016

- 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 548–572 (2016), https://doi.org/10.1007/978-3-662-53887-6_20
10. Martin, D.P., Montanaro, A., Oswald, E., Shepherd, D.J.: Quantum key search with side channel advice. IACR Cryptology ePrint Archive 2017, 171 (2017), http://eprint.iacr.org/2017/171
11. 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.) Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 313–337. Springer (2015), https://doi.org/10.1007/978-3-662-48800-3_13
12. Poussier, R., Grosso, V., Standaert, F.: Comparing approaches to rank estimation for side-channel security evaluations. In: Homma, N., Medwed, M. (eds.) Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers. Lecture Notes in Computer Science, vol. 9514, pp. 125–142. Springer (2015), https://doi.org/10.1007/978-3-319-31271-2_8
13. Poussier, R., Standaert, F., Grosso, V.: Simple key enumeration (and rank estimation) using histograms: An integrated approach. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9813, pp. 61–81. Springer (2016), https://doi.org/10.1007/978-3-662-53140-2_4
14. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7707, pp. 390–406. Springer (2012), https://doi.org/10.1007/978-3-642-35999-6_25
15. Veyrat-Charvillon, N., Gérard, B., Standaert, F.: Security evaluations beyond computing power. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 126–141. Springer (2013), https://doi.org/10.1007/978-3-642-38348-9_8
16. 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.) Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8968, pp. 215–232. Springer (2014), https://doi.org/10.1007/978-3-319-16763-3_13