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

Vincent Grosso

Radboud University Nijmegen, Digital Security Group, The Netherlands.

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.

Keywords: Side-channel analysis · Evaluation · Security assessment

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. This kind of attacks has been used to mount some practical attacks against real devices [LYS+15]. Hence, cryptographic algorithms required secure implementations.

To defeat side-channel attacks, cryptographic implementations should embedded appropriate countermeasures. The security margins that the countermeasures offer should be tested. For this reason, 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 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 [BLvV15, GGP+15, MOOS15, VGS13]. 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 [GGP+15] 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). Remark that our method offers a trade-off between efficiency and tightness of the result. That is a new feature for large key evaluation as the only efficient solution of Choudary and Popescu [CP17] can not tighten the bounds provided.

Finally, by applying a similar idea as Poussier et al. [PSG16], we transform our key rank algorithm to a key enumeration algorithm. This key enumeration algorithm could be useful when the CHES 2016 enumeration algorithm requires too much memory.

2 Background

2.1 Side-channel attacks

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 η-bit master key. An adversary using a divide-and-conquer strategy will split this key into ν subkeys of (for simplicity equally sized) κ bits. 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 does not obtain probabilities, but scores it could either use a Bayesian extension [VGRS12] or use direct results [CPS16].

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 [BKM+15, DW17, MMOS17, MOOS15, PSG16, VGRS12]. All these algorithms have been tested in a symmetric key setting and provide efficient solution.

The algorithms proposed in [BKM+15, MOOS15, PSG16] 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 side-channel attack, i.e. how many keys the attacker needs to test in the recombination phase before she recovers the actual key. 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.

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

Where # stands for the cardinality of the set.

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 [BLvV15, GGP$^+$15, MOOS15, VGS13]. Among these solutions only [VGS13] uses the probabilities and thus can compute the actual rank, if enough time and memory are given to the program. Remark this time could be (too) long and memory requirement could be (too) large. For that reason, several rank estimation algorithms have been proposed [BLvV15, GGP$^+$15, MOOS15]. All these proposals share the same step that introduces error: they map the probabilities to integers (see [PGS15] 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.

Definition 2 (Tightness)**.** We define the tightness of an estimation as the logarithm of the ratio between the upper and lower bound for the rank $\log_2\left(\frac{rank\_upper\_bound}{rank\_lower\_bound}\right)$. When we manage to compute the exact rank of the key the tightness is 0. The tightness of an estimation gives an intuition on the accuracy of security margins.

These rank estimation algorithms are based on samples, they try one 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 [VGS13].

Some other solutions exist to evaluate the security of a device that can be faster and adaptable for large keys [DFS15, YEM14]. These solutions are based on metrics. However, the solutions based on metric could misestimate the actual computational power to recover a key, as pointed out in [PGS15].

Another recent approach to evaluate the security for large keys has been presented in [CP17]. The authors propose instead of using the rank of the key to use the expected value of the key rank. They manage to compute quite efficient bound on the expected value of the key rank. Unfortunately, this result seems disconnected to the rank/enumeration problem. In [MMOS16] the authors show the limitations to only use the expected value of the key rank. Moreover, the gap between the upper and lower bound can not be reduced. As the gap is between 5 and 10 bits it can be seen as too large for some cases.

In [CP17] a list of pros and cons for some method to evaluate security margins are give. We list in Table 1 some pros and cons for key ranking algorithms.

The tightness of rank estimation depends on the precision parameters for algorithms that have such parameter ([BLvV15, GGP$^+$15, MOOS15]). But also from the rank of the key and the distribution of probabilities. Thus giving formulas to compare time efficiency of different solutions in function of their tightness is difficult.

2.4 The histogram solution

Since our solution is based on the Glowacz et al. solution [GGP$^+$15], 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 $\Pr[k_1, k_2|SCI] = \Pr[k_1|SCI] \times \Pr[k_2|SCI]$. To use the FSE'15 solution we

Table 1: Comparison of key rank estimation algorithm

Method	Pros	Cons
Eurocrypt'13 [VGS13]	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro [BLvV15]	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
FSE'15 [GGP+15]	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
Asiacrypt' 15 [MOOS15]	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
CT-RSA'17 [DW17]	Efficient solution	Loose bound
CHES'17 [CP17]	Really fast even for large key	Expected value of the rank, not rank estimation

need an additive relation. For that reason, we use the logarithm of probabilities. Hence, 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. From now the log probability is assimilated to the center of the bin. The distance between the log probability and the center of the bin is half of the bin width.

3. *convolution of histograms*: the convolution of histograms gives us the distribution of the combination of the probabilities of different combination of subkeys. In order to have some meaningful probability at the end the histograms should have bins of equal size. 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: the list of histogram score for each subkey (hi(subkey,:))
%b: the 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 [GGP+15] 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 2. 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 2: Probabilities of the different 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.

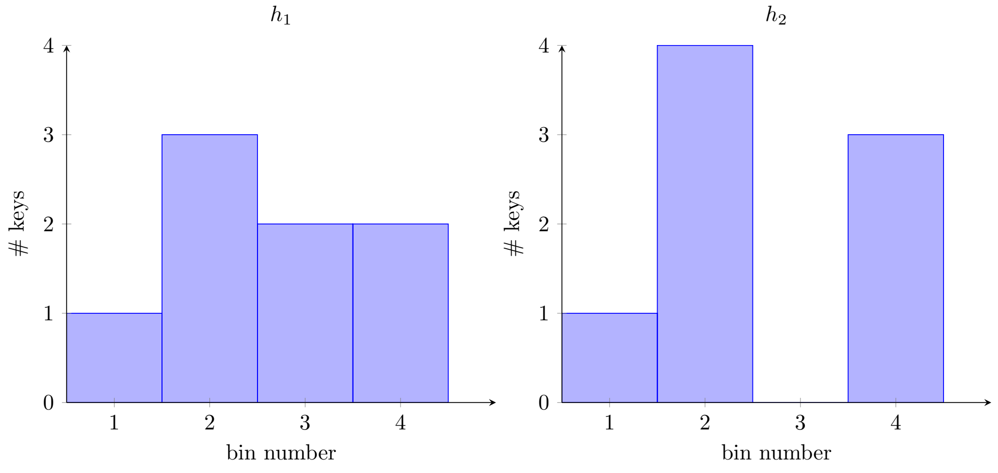

Figure 1: The histograms for the two subkeys.

h_1 is the histogram 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 2, the bin 1 should correspond to the

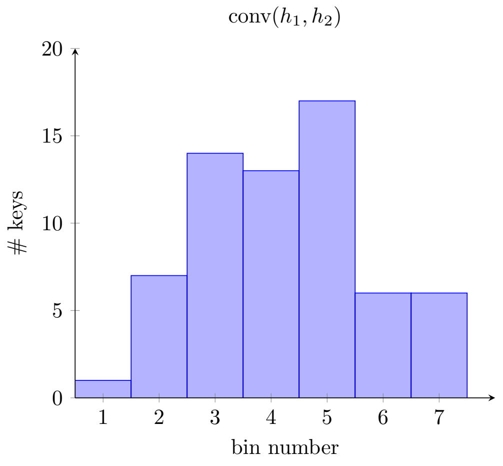

Figure 2: The convolution result.

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.

The most expensive part of such an algorithm is the part 3, or loop for of line 10 in listing 1. 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 [GGP+15].

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 20 in listing 2).

Listing 2: Matlab implementation of our solution.

```
function [mini,maxi] = rank(hi)
% Inputs:
%hi: the list of histogram score for each subkey (hi(subkey,:))
%b: the 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);
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 4.

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 2.

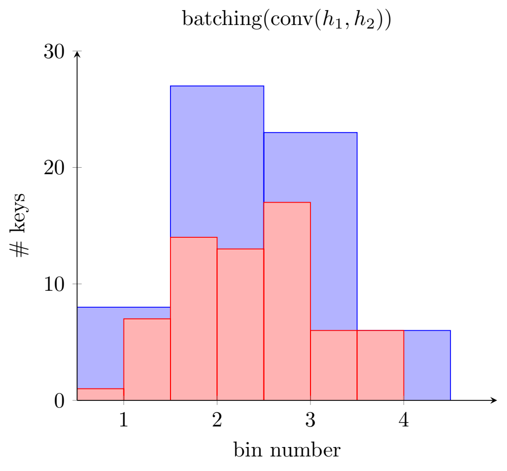

Figure 3: The batched result.

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.

$$\begin{tikzcd}
h_2 \arrow[r, "convolution"'] & H_1 \arrow[r, "convolution"'] & H_2 \arrow[r, "convolution"'] & H_3 \\
h_1 \arrow[ur] & h_3 \arrow[ur] & h_4 \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
h_1 \arrow[dr] & & h_2 \arrow[dl, "conv."'] & & h_3 \arrow[dr] & & h_4 \arrow[dl, "conv."'] \\
& H \arrow[d, "batching"] & & & & H \arrow[d, "batching"] & \\
& H_2 \arrow[drr] & & & & H_2 \arrow[dll, "convolution"'] & \\
& & & H \arrow[d, "batching"] & & & \\
& & & H_2 & & &
\end{tikzcd}$$

Figure 4: Representation of the FSE'15 solution (left) and ours (right).

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 convolution in level 1 of the tree (right part of figure 4) 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 4 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.

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 goes from real numbers to 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:

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

By iterating the error propagation we obtain error for our method. Some values of errors for different numbers of subkeys are displayed in Table 3. Remark the error can be calculated 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 $2^{\nu+1}\epsilon$

Table 3: Error evolution for our method, we assume that initial histogram have bin width of 2ϵ.

ν	bin width final histogram	error
1	2ϵ	ϵ
2	4ϵ	3ϵ
4	8ϵ	8ϵ
8	16ϵ	20ϵ
16	32ϵ	48ϵ
32	64ϵ	112ϵ
64	128ϵ	256ϵ

We do not give error bounds as a number of bins as for FSE as the size of our bins are large calculate the lower and upper bins from the log probability of the key $\pm$ error give a better result than starting from the bin center of the log probability $\pm$ number of margin bins.

Remark that such approach to calculate the error bound could be applied to the case of non equally sized histograms. In such a case we can have more bins close to the actual bin of the key. This may lead to tighter bounds on the rank.

3.4 Non power of 2 cases

If the number of subkeys is not a power of two our first convolution step (line 10 in listing 2) should be adapted as described in listing 3.

Listing 3: Matlab implementation for non power of 2.

```
function [mini,maxi] = rank3(hi, b)
% Inputs:
%hi: the matrix of histogram score for each subkey (hi(subkey,:)
    )
[dim,shi]=size(hi);
tmp=floor(log2(dim))
H2=cell(ceil(log2(dim)),dim/2);
for i=2:2:dim-tmp
    H=conv(hi(i-1,:),hi(i,:));
    H2{1,i/2}=[H(2:2:length(H)),0]+H(1:2:length(H));
end
for i=dim-tmp+1:dim
    H2{1,i/2}=hi(i,2:2:shi)+hi(i,1:2:shi);
end
```

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.

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 example against Mifare DESFire [OP11].

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. In particular, we want to verify the good performance of our method when the number of subkeys increases.
- *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. Note that the precision and the tightness are different notions. The precision is a parameter of the solution, while the tightness is obtained from the output of the algorithm and the tightness depends on several parameters (e.g. rank, leakage function, precision ...).
- *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. As we can observe only one deterministic value per subkey, the number of trace and noise are directly linked, as more measurement will allow dividing the noise.
- *The size of the subkeys.* For our experiments, we target 8-bit subkeys. The size of subkeys has an impact mostly on the creation of histogram as we need to enumerate all subkeys. But this could be done as a side-channel attack is possible on the subkeys.

All these parameters and the values used are summarized in Table 4.

Table 4: Parameters of our simulation and their principal values

Parameter	Values
number of subkeys	16,32,64,128,256,512,1024
precision	$2^8, 2^{12}, 2^{16}$
leakage function	Identity, Hamming weight
noise variance	$2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, \dots, 2^{-8}$
size of subkeys	8-bit

4.1 Same precision

First, we compare our method with the FSE'15 method for the same number of bins. In a first time, we just check that our method gives loose bounds. Indeed for the results, we get the max rank of our method is higher than the max rank of FSE and the min of our method is lower than the min of FSE'15. This is a sanity check for our method. Result are display in Tables 5 and 6.

Table 5: Result for our method versus FSE'15 for 16 subkeys, 2^{12} bins, identity leakage.

	σ	8	4	2	1	0.5	0.25
FSE	$\log_2(max)$	127.231	126.966	119.386	108.297	105	88.2963
	$\log_2(min)$	127.224	126.956	119.332	108.159	104.778	87.6903
Ours	$\log_2(max)$	127.234	126.968	119.426	108.341	105.16	88.4205
	$\log_2(min)$	127.211	126.929	119.222	107.815	104.31	85.9786
	σ	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625
FSE	$\log_2(max)$	69.405	58.2386	38.3047	25.8204	5.2854	0
	$\log_2(min)$	68.6052	57.6567	37.4963	25.2285	4.58496	0
Ours	$\log_2(max)$	69.6631	58.7153	38.8743	26.1853	5.39232	0
	$\log_2(min)$	66.3938	56.4592	35.6711	23.8557	1	0

Table 6: Result for our method versus FSE'15 for 16 subkeys, 2^{12} bins, hamming weight leakage.

	σ	8	4	2	1	0.5	0.25
FSE	$\log_2(max)$	125.679	124.641	120.576	113.205	95.0947	83.7449
	$\log_2(min)$	125.652	124.607	120.484	113.016	94.3633	83.2428
Ours	$\log_2(max)$	125.698	124.658	120.66	113.3	95.5892	83.7449
	$\log_2(min)$	125.596	124.526	120.308	112.577	92.6648	82.7093
	σ	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625
FSE	$\log_2(max)$	78.1395	86.2688	87.72	92.8492	78.5248	86.0761
	$\log_2(min)$	0	0	0	0	0	0
Ours	$\log_2(max)$	78.1395	86.2688	87.72	92.8492	78.5248	86.0761
	$\log_2(min)$	0	0	0	0	0	0

For the hamming weight leakage (the results are display in Table 6), we can see that the methods do not provide tight bounds. This comes from the leakage function and the fact we can observe only one value per subkey. That means if there exist k and k' such that the leakage function gives the same output, we cannot differentiate the two subkeys. Thus the two subkeys will have the same probability. Typically if the 16 subkeys have a Hamming weight of 4 we have $\binom{8}{4}^{16} \simeq 2^{96}$ keys that we cannot differentiate. In [MOOS15] they said that in such case the key should be considered as the first one tested. It makes some senses if we want to asset security, but it seems to reduce highly the security margins in the example of memory Hamming weight leakage. In order to avoid such problem we next just use the identity leakage. While the Hamming weight is a good approximation in general of the leakage behavior more precise model could increase the success of an attack.

Next, 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 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.

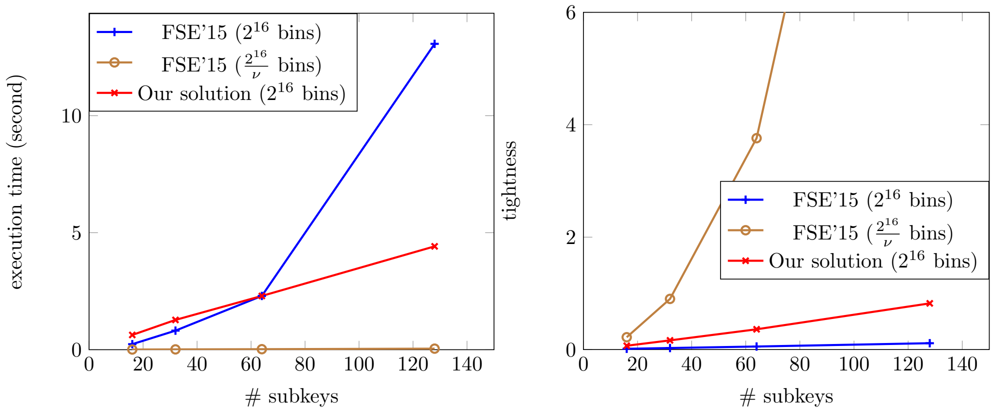

Figure 5: Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for different sizes of keys (16 bits of precision) and an SNR of 8.

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 6.

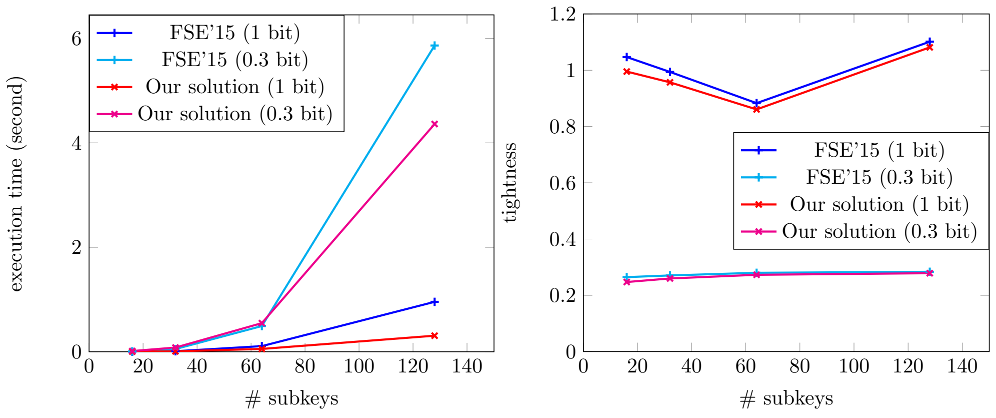

Figure 6: Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for similar tightness.

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 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. [GGP+15] suggest to use Chinese remainder theorem. Another issue of using doubles instead of integers is that above 2^{53} not all integers are represented and thus some error could appear. In the same time, the CRT requires to compute several times the convolutions and thus impact the efficiency of the solutions.

To override these issues we implement our solution using a big integer library: the NTL library.[1] 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. The results are reported in the figure 7.

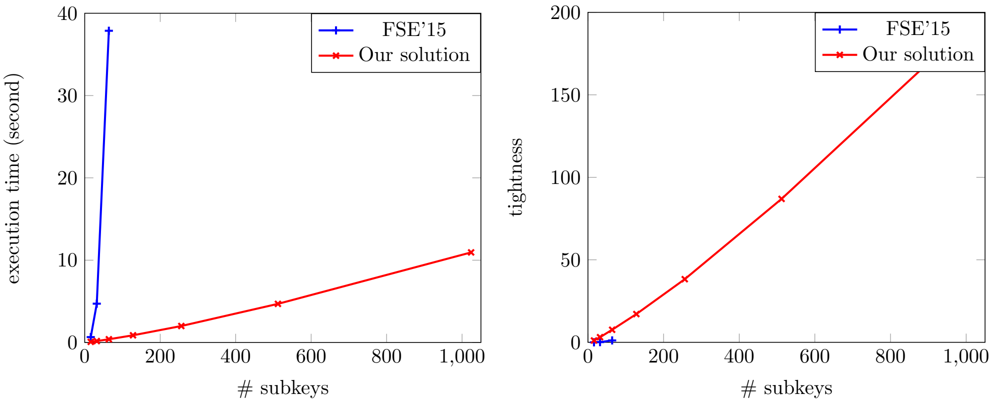

Figure 7: Execution time (left) and tightness of the bound (right) of NTL convolutions for histogram using FSE'15 method and ours, starting with histograms of 2^{12} bins and SNR 8.

We can see that for the NTL implementation our method is more efficient even for small key size. The efficiency gap is probably due to the fact that the NTL implements several techniques for the convolution by using large coefficients and a small number of bins the NTL select to use the Schoenhage and Strassen [SS71] rather than the CRT technique because it is more efficient in that case. Another remark is that for the FSE method the NTL seems less efficient than the Matlab implementation here also this is probably due to the underlying method to perform convolutions.

Finally, we can see that our method offer an efficient manner to calculate rank even for the case when the key is divided into 1024 subkeys.

4.4 Comparison with CHES 2017

At CHES 2017 Choudary and Popescu present an "impressively fast, scalable and tight security evaluation tools" [CP17]. Note that their tool does not calculate the rank of the key but the expected value of the rank. As pointed out in [MMOS16] it is not clear how

[1]. http://www.shoup.net/ntl/

to evaluate the power computation required to recover the key from the expected value of the rank.

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 8.

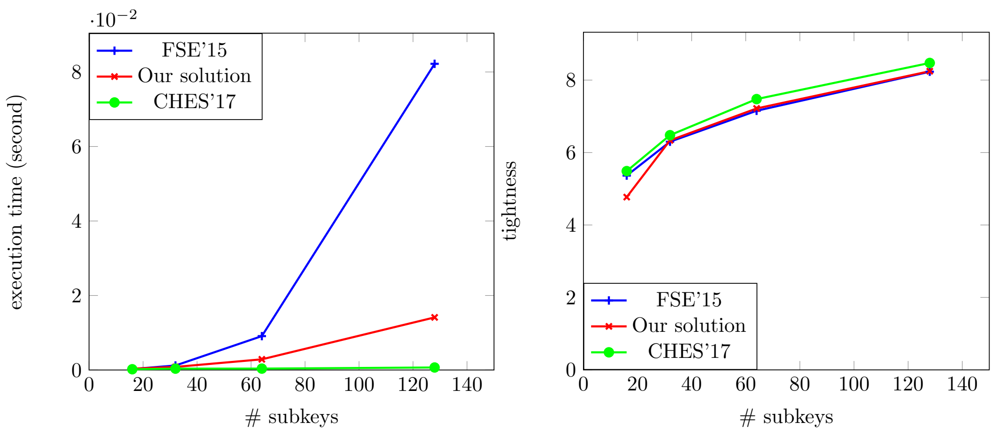

Figure 8: Execution time (left) and tightness of the bound (right) of Matlab implementations of the FSE'15 solution and ours for similar tightness as CHES'17.

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.

In CHES'17 Choudary et al. leave some figures as references for large key evaluation. For that, we use our C implementation describe in Subsection 4.3. To obtain tighter bounds. We choose to merge list of the 1024 lists of 8-bit probability. Thus we have 512 lists of 16-bit key candidates. We then use our technique with 2^{13} bins. In such setting, we manage to obtain bound of tightness 30 bits in 48 seconds for average rank 2^{5500}. Our method is less efficient/precise but we calculate different things.

5 Key enumeration

We can apply similar idea as the backtracking used in [PSG16]. The idea of this paper is to first perform a construction step that is the three first steps of rank estimation. Then a backtracking of the solution is performed to recover the key candidates of each bin. Our technique speed up the construction phase of a solution like [PSG16]. In general, this step is negligible in such solution. So our solution seems to do not brings any major speed up of enumeration algorithms.

In Algorithm 1 we describe our solution the adaptation of “decompose bin" of [PSG16]. We need to adapt the algorithm to the shape of our tree versus the tree they used (see Figure 4). The modifications are the different conditions the if and the elseif (lines 1 and 4). The second adaptation came from the fact we have batching to take into account. The modifications for this part are in the else condition (lines 6).

Algorithm 1 Decompose bin

```
Require: $H$: the structure containing all the histograms output by the construction step
    $ch$ (current hist): the index of the current histogram of $H$ we target
    $x_{bin}$: The bin index of $H_{ch}$ we want to decompose.
Ensure: $k_f$: A data structure containing the candidate keys.
 1: if $ch = 0$ then
 2:     $k_f \leftarrow get(H_0, x_bin)$
 3:     $process(k_f)$
 4: else if $ch < \nu$ then
 5:     $k_f \leftarrow get(H_{ch}, x_bin)$
 6: else
 7:     $x \leftarrow sizeof(right\_son\_of(H_{ch}))$
 8:     while $(x \geq 0)$ and $(x + sizeof(left\_son\_of(H_{ch})) \geq 2 \times x_{bin})$ do
 9:         if $right\_son\_of(H_{ch})(x) > 0)$ and
                $(left\_son\_of(H_{ch})(2 \times x_{bin} - x) > 0$ or
                $left\_son\_of(H_{ch})(2 \times x_{bin} - x + 1) >)0$ then
10:             $Decomposebin(right\_son\_of(H_{ch}), x, H, kf)$
11:             if $left\_son\_of(H_{ch})(2 \times x_{bin} - x) > 0$ then
12:                 $Decomposebin(left\_son\_of(H_{ch}), 2 \times x_{bin} - x, H, kf)$
13:             end if
14:             if $left\_son\_of(H_{ch})(2 \times x_{bin} - x + 1) > 0$ then
15:                 $Decomposebin(left\_son\_of(H_{ch}), 2 \times x_{bin} - x + 1, H, kf)$
16:             end if
17:         end if
18:         $x \leftarrow x - 1$
19:     end while
20: end if
21: return
```

Our enumeration algorithm has an advantage over the previous Poussier et al. when memory needed to store histograms is too large.

6 Conclusion

This paper presents a simple trick to reduce the cost of rank estimation for a large number of subkeys based on the rank estimation of [GGP+15]. 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. Another advantage of keeping the number of bin "small" with large coefficient is that the method for convolution used could vary and for example some techniques as the Schoenhage and Strassen [SS71] could be more efficient than the CRT method for such a case. 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.

References

[BKM+15] Andrey Bogdanov, Ilya Kizhvatov, Kamran Manzoor, Elmar Tischhauser, and Marc Witteman. Fast and memory-efficient key recovery in side-channel attacks. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 310–327. Springer, 2015.

[BLvV15] Daniel J. Bernstein, Tanja Lange, and Christine van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. *IACR Cryptology ePrint Archive*, 2015:221, 2015.

[CP17] Marios O. Choudary and P. G. Popescu. Back to massey: Impressively fast, scalable and tight security evaluation tools. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 367–386. Springer, 2017.

[CPS16] Marios O. Choudary, Romain Poussier, and François-Xavier Standaert. Score-based vs. probability-based enumeration - A cautionary note. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, volume 10095 of *Lecture Notes in Computer Science*, pages 137–152, 2016.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

[DW17] Liron David and Avishai Wool. A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2017.

[GGP+15] Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2015.

[LYS+15] Junrong Liu, Yu Yu, François-Xavier Standaert, Zheng Guo, Dawu Gu, Wei Sun, Yijie Ge, and Xinjun Xie. Small tweaks do not help: Differential power analysis of MILENAGE implementations in 3G/4G USIM cards. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, volume 9326 of *Lecture Notes in Computer Science*, pages 468–480. Springer, 2015.

[MMOS16] Daniel P. Martin, Luke Mather, Elisabeth Oswald, and Martijn Stam. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *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*, volume 10031 of *Lecture Notes in Computer Science*, pages 548–572, 2016.

[MMOS17] Daniel P. Martin, Ashley Montanaro, Elisabeth Oswald, and Dan J. Shepherd. Quantum key search with side channel advice. *IACR Cryptology ePrint Archive*, 2017:171, 2017.

[MOOS15] Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In Tetsu Iwata and Jung Hee Cheon, editors, *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*, volume 9453 of *Lecture Notes in Computer Science*, pages 313–337. Springer, 2015.

[OP11] David Oswald and Christof Paar. Breaking mifare DESFire MF3ICD40: power analysis and templates in the real world. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2011.

[PGS15] Romain Poussier, Vincent Grosso, and François-Xavier Standaert. Comparing approaches to rank estimation for side-channel security evaluations. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2015.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[VGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2012.

[VGS13] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of*

Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, volume 7881 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2013.

[YEM14] Xin Ye, Thomas Eisenbarth, and William Martin. Bounded, yet sufficient? how to determine whether limited side channel information enables key recovery. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2014.