Fast and Memory-Efficient Key Recovery in Side-Channel Attacks

Andrey Bogdanov[1], Ilya Kizhvatov[2], Kamran Manzoor[1,2,3],
Elmar Tischhauser[1], and Marc Witteman[2]

[1] Department of Applied Mathematics and Computer Science
Technical University of Denmark, Denmark
{anbog,ewti}@dtu.dk
[2] Riscure B.V., The Netherlands
{kizhvatov,witteman}@riscure.com
[3] Aalto University, Finland
kamran.manzoor@aalto.fi

Abstract. Side-channel attacks are powerful techniques to attack implementations of cryptographic algorithms by observing its physical parameters such as power consumption and electromagnetic radiation that are modulated by the secret state. Most side-channel attacks are of divide-and-conquer nature, that is, they yield a ranked list of secret key chunks, e.g., the subkey bytes in AES. The problem of the key recovery is then to find the correct combined key.

An optimal key enumeration algorithm (OKEA) was proposed by Charvillon et al at SAC'12. Given the ranked key chunks together with their probabilities, this algorithm outputs the full combined keys in the optimal order – from more likely to less likely ones. OKEA uses plenty of memory by its nature though, which limits its practical efficiency. Especially in the cases where the side-channel traces are noisy, the memory and running time requirements to find the right key can be prohibitively high.

To tackle this problem, we propose a score-based key enumeration algorithm (SKEA). Though it is suboptimal in terms of the output order of cadidate combined keys, SKEA's memory and running time requirements are more practical than those of OKEA. We verify the advantage at the example of a DPA attack on an 8-bit embedded software implementation of AES-128. We vary the number of traces available to the adversary and report a significant increase in the success rate of the key recovery due to SKEA when compared to OKEA, within practical limitations on time and memory. We also compare SKEA to the probabilistic key enumeration algorithm (PKEA) by Meier and Staffelbach and show its practical superiority in this case.

SKEA is efficiently parallelizable. We propose a high-performance solution for the entire conquer stage of side-channel attacks that includes SKEA and the subsequent full key testing, using AES-NI on Haswell Intel CPUs.

1 Introduction

In a nutshell. Side-channel attacks are powerful techniques to extract cryptographic keys from secure chips. With a few exceptions, they are divide-and-conquer attacks recovering individual small chunks of the key. For every chunk of the key, the attack's divide phase called a *distinguisher* will assign a ranking value to every possible candidate. Depending on the attack technique, this value can be a correlation coefficient or a probability value. The output of the distinguisher is a set of lists of all key chunk candidates and their ranking values.

If the attacker can perform a sufficient number of measurements to reduce noise, the *conquer* phase is trivial. The correct full key is obtained by taking the top-ranking candidate. However, this is rare in practical settings with low signal-to-noise ratio due to target complexity and countermeasures, and limited time to perform the attack. In such practical settings, the correct key chunks are not likely to be ranked on the top of the list, but somewhere close to the top. Hence, to recover the full key, the attacker needs to perform the search over different combinations of key chunk candidates. By taking into account the ranking values coming from the distinguisher, the search can be made more efficient. The problem is known as *key enumeration*. More generally, besides key enumeration, the conquer phase also includes key verification.

Related work and motivation. Though most research in side-channel attacks focuses on the divide phase, there are several papers tackling the issue of key enumeration. The most recent line of relevant research is the optimal key enumeration algorithm (OKEA) proposed Charvillon et al. in SAC'12 [14, 15]. It is optimal with respect to the order of the full keys by their probabilities. OKEA is superior to the very similar algorithm of Pan et al. from SAC'10 [10] in terms of memory use, and to the probabilistic key enumeration algorithm (PKEA) suggested by Meier and Staffelbach in EUROCRYPT'91 [9] (and well applicable in the context of DPA and DFA) in terms of the amount of candidates tried. A dedicated solution for the case of an attack recovering individual bits was proposed by Dichtl in COSADE'10 [3]. The papers [8, 16] include solutions to tackle the conquer phase as well.

The optimal key enumeration algorithm from [15] has some practical limitations though. First, for practical cases, it consumes a lot of memory and its performance is hampered by random memory access pattern. Second, it is difficult to efficiently parallelize on a multi-core CPU[1]. Thus, we are more interested in the practical optimality, i.e., an optimality in terms of how much time on average it takes to find the key on a modern CPU. Previous papers lack a systematic comparison with respect to this metric.

[1] We focus on a common non-specialized platform, namely, on a workstation with a multi-core CPU. We are fully aware that dedicated platforms, in particular GPUs and FPGAs, may bring faster smart brute force solutions; they are out of scope of this work.

In an independent work [7], a parallelizable score-based key enumeration algorithm is presented. It casts the key enumeration problem as a solution to a multi-dimensional integer knapsack, which is achieved by listing all valid paths in the respective graph representation. The algorithm is shown to be more efficient than OKEA. The approach also provides efficient key rank estimation by counting paths in the graph.

Quality versus speed. Two aspects are of great importance when searching for a key: *quality* and *speed*. With *quality* we mean the relation between candidate generation and the correctness probability of a key candidate. A brute force algorithm that tries every key candidate in the plain numerical order is likely to have a bad quality. This is because probable key candidates will not get precedence over unprobable candidates. With *speed* we mean how fast subsequent key candidates are generated. Typically the key verification can be done with optimized hardware or software implementations of the cryptographic algorithm. Ideally, the enumeration should be faster than the verification to benefit from the optimization of the cryptographic implementation.

The overall performance of the approach, i.e. *the time it takes to find the correct key*, depends on both quality and speed. In this paper we look at both factors and find that a high quality may reduce speed and suboptimality may minimize the search time.

Our contributions. The contributions of this paper are as follows. First, we propose a new key enumeration algorithm, namely, the score-based key enumeration algorithm (SKEA), that can find the full key in practice faster than OKEA and can be efficiently parallelized. Second, we provide a comprehensive comparison of OKEA, PKEA and SKEA in terms of the practical runtime metric for DPA on an embedded 8-bit software target. Finally, we propose a full solution for the conquer stage (including key enumeration and testing) for AES that includes an efficient key verification implementation using AES-NI extensions on Haswell CPUs.

Thus, our work aims to bridge the gap between theory and practice in key enumeration and puts forward the importance of the practical metrics for estimating implementation attack complexity.

Part of this work has been first presented in [6] and [1].

2 Background on key enumeration algorithms

Most side-channel attacks are based on a divide-and-conquer approach in which a side-channel distinguisher is used to obtain ranking information about parts (chunks) of the key (“divide”) which is then used to determine candidates for the complete key (“conquer”). This complete key is the result of a function, e.g. concatenation, of all the key parts. A key part candidate is a possible value of a key part that is chosen because the attack suggests a good probability for that value to be correct.

For instance, in a side-channel attack on the S-box step of a block cipher, such a key part comprises all key bits corresponding to a single S-box. The complete key recovered by the attack would be the concatenation of all such key parts in one round.

The problem of finding (candidates for) the complete key based on the observed side-channel information for the key parts is called the *key enumeration problem*: For each of the m key parts, we are given a list of key part candidates, each of which has a certain associated side-channel information (e.g., leakage). We are then to enumerate candidates for the complete key, ideally in order of likelihood according to the likelihood of the individual key part candidates.

Several key enumeration algorithms have been proposed, among them the classic probabilistic algorithm (PKEA) due to Meier and Staffelbach [9], peak distribution analysis by Pan et al. [10], and the recent optimal key enumeration (OKEA) by Veyrat-Charvillon et al. from SAC 2012 [15].

Ranking vs. likelihood. Both PKEA and OKEA require the discrete probability distribution of each subkey part to be known and computable by the attacker. In profiled side-channel attacks, the training phase allows the attacker to rank the key part candidates k for each of the m key parts according to their actual probabilities $\Pr(k \mid L_i)$, with L_i denoting the complete available side-channel information for the i-th key part. In non-profiled attacks, the ranking of key part candidates is performed according to certain statistics extracted from the side-channel information, which might not reflect their actual likelihood. In [15], a method called *Bayesian extension* has been proposed in order to convert the ranking statistics into an equivalent probability value based on a certain leakage model. It essentially does a Bayesian model comparison to obtain the conditional probabilities $\Pr(k \mid L_i)$ from the probabilities $\Pr(L_i \mid k)$ which are obtained according to the leakage model.

2.1 Probabilistic key enumeration (PKEA)

At EUROCRYPT 1991, Meier and Staffelbach [9] describe a probabilistic key enumeration algorithm which we denote PKEA. In their work, the attacker is assumed to have no knowledge about the key part distributions, but is able to sample from them. The algorithm then simply samples key parts according to their individual distributions and tests the resulting complete key candidates.

Besides its simplicity, one of its main advantages is that it only requires a small constant amount of memory. On the other hand, the most likely key candidates will occur many times due to the stateless nature of this algorithm, which leads to unnecessary work duplication.

2.2 Optimal key enumeration (OKEA)

At SAC 2012, Veyrat-Charvillon et al. proposed a deterministic key enumeration algorithm which guarantees to output the key candidates in nondecreasing order of their likelihood [15]. Due to this property, it is referred to as an optimal key enumeration algorithm, or OKEA.

This algorithm is based on the idea that if the key part candidates for the m key parts are given in decreasing order of their probability, the m-dimensional space of all complete key candidates can be enumerated in order of their likelihood by only keeping track of the next most likely candidate in each dimension (this set of key part candidates is called the *Frontier set*). By pruning already enumerated complete key candidates and following a recursive decomposition for higher dimensions than two, this algorithm generates key candidates in optimal order without having to keep all possible key part combinations in memory at any time.

While constituting a significant improvement over both PKEA and the naive enumeration approach, OKEA is usually memory-bound, which means that even though keys are enumerated in optimal order, the memory required to keep track of the (nested) Frontier sets can be prohibitive in practice before the correct key is found, especially for higher dimensions such as $m = 16$ for attacks on the AES. Furthermore, the way the recursive decomposition is performed and the intermediate results are recombined also makes effective parallelization of OKEA difficult.

3 Our score-based algorithm (SKEA)

In this section, we introduce SKEA, our proposed score-based algorithm for the key enumeration problem. Like OKEA, it is a deterministic algorithm. Instead of guaranteeing an optimal enumeration order, it is intended to be optimized for practical applicability by only employing a fixed maximum amount of memory (as opposed to OKEA's dynamicly increasing memory usage) and by being easily parallelizable.

3.1 Score concepts

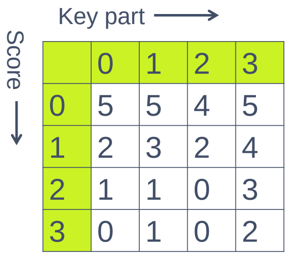
SKEA works with *scores* which are assigned to each key part candidate based on the likelihood that it will be part of the correct complete key according to the available side-channel information.

Scores are integer values, and can be derived for instance from a correlation value in a side-channel analysis attack, or the number of faults that match with a candidate key part value in differential fault attack. SKEA can deal with scores assigned from an arbitrary integer interval $[s_{\min}, s_{\max}]$, and the size of this interval will typically be chosen according to the required precision, since a larger interval means more distinctive possible scores.[2]

[2] For a standard power analysis attack on an 8-bit S-box in AES, computing the score as $\lfloor 50\,|\rho| + 0.5 \rfloor$ for a correlation coefficient of ρ, resulting in an interval size of 50, was found to yield good results (see Sect. 5).

We furthermore define the *cumulated score* as the sum of all the key part scores comprising one complete key. SKEA generates candidate keys by descending cumulated scores. It starts by generating the key with the largest possible cumulated score $S_{\max}$, and proceeds by finding all keys with a score equal to $S_{\max} - 1$, and so forth. This property guarantees a certain quality, i.e. good key candidates will be enumerated earlier than worse ones. Note that candidates with the same score are generated in no particular order, so a lack of precision in scores will lead to some decrease of quality.

An example with $m = 4$. For simplicity, we will explain the score concept for $m = 4$ key parts with 4 candidates each (corresponding to 4 key parts of 2 bits each). This will also serve as a running example to illustrate the SKEA algorithm. Figure 1 shows a simple example of a score table. The key consists of columns representing 4 key parts that can have only 4 values. The cells in the columns contain the sorted scores. Row 0 shows the best scores for all key parts, while row 3 shows the worst scores. A complete key candidate consists of the concatenation of selected cells for all collumns, having a cumulated score equal to the sum of the individual scores for each key part. We directly see that the best cumulated score would be 19, resulting from the sequence $\{0, 0, 0, 0\}$, while the worst cumulated score would be 3, resulting from the sequence $\{3, 3, 3, 3\}$.


Key part →
Score ↓

	0	1	2	3
0	5	5	4	5
1	2	3	2	4
2	1	1	0	3
3	0	1	0	2

Fig. 1. Example of a score table for $m = 4$ and a 2-bit S-box.

3.2 Enumerating score paths

SKEA now makes use of the sorted score table in order to enumerate key candidates in order of decreasing cumulated scores. We refer to a combination of key part candidates with cumulated score equal to s to a possible *score path* for score s. Given a best possible cumulated score of $S_{\max}$, we then must find all possible score paths leading to a cumulated score of $S_{\max}$, followed by $S_{\max} - 1$, and so on.

In order to find valid score paths conforming to a certain cumulated score, SKEA makes use of a simple backtracking strategy. This algorithm essentially performs a depth-first search, which is efficient because impossible paths can be pruned early. The pruning is controlled by the score that must be reached for the solution to be accepted.

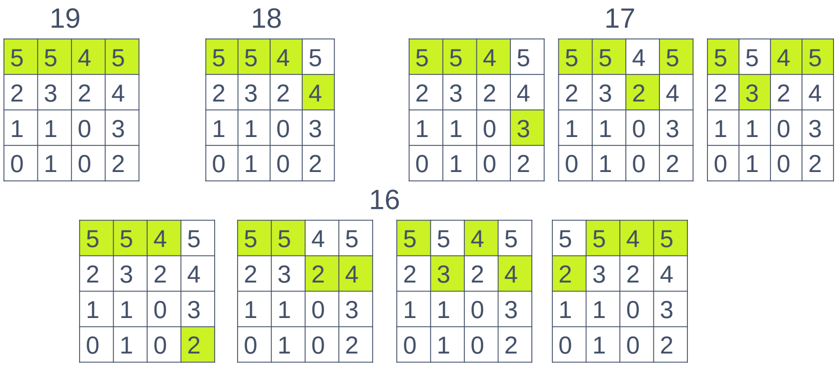


Fig. 2. Possible score paths conforming to cumulated scores of 19, 18, 17 and 16.

We continue our toy example with $m = 4$ and 2-bit key parts from Sect. 3.1 by listing all possible score paths s for cumulated scores of 19, 18, 17, and 16 in Fig. 2.

3.3 Efficiently eliminating incompatible score paths

To achieve a fast decision process during the backtracking, SKEA makes use of precomputed tables containing possible ranges of key part selection indices that may lead to score paths with the desired cumulated score. Suppose we are given a score table $S[i][k]$ containing the scores of candidate k for the i-th key part.

Cumulated minimum and maximum scores. First we compute tables for minimal and maximal cumulated scores that can be reached by completing a path to the right:

$$\mathrm{cMin}[i] = \min_k \sum_{j=i}^{m-1} S[j][k], \quad 0 \le i < m,$$

$$\mathrm{cMax}[i] = \max_k \sum_{j=i}^{m-1} S[j][k], \quad 0 \le i < m.$$

When completing a score path from left to right, the cumulated minimum and maximum score table gives the minimum and maximum score that can be added by including cells from the current column, and to the right.

For our running example, these tables are given in Fig. 3. They show that by adding only the rightmost column, a minimum of 2, and a maximum of 5 can be added to the score. Likewise, by adding the 2nd, 3rd, and 4th columns a minimum of 3, and a maximum of 14 can be added to the cumulated score.

cMin			
3	3	2	2

cMax			
19	14	9	5

Fig. 3. Minimum and maximum cumulated scores

Key part index limits. Using the cumulated minimum and maximum tables, we create two additional tables that hold the indices for completing the path successfully for a certain desired cumulated score, when proceeding from a score path up to a given key part index:

$$\begin{aligned} \mathrm{iMin}[i][s] &= \arg\min_k \left(S[i][k] + \mathrm{cMin}[i+1]\right) \geq s, \\ \mathrm{iMax}[i][s] &= \arg\max_k \left(S[i][k] + \mathrm{cMax}[i+1]\right) \leq s, \\ &\quad 0 \leq i < m, \mathrm{cMin[i]} \leq s \leq \mathrm{cMax[i]}, \mathrm{cMin}[m] = \mathrm{cMax}[m] = 0. \end{aligned}$$

Cumulated score ↓

Key part 0

	Min	Max
19	0	0
18	0	0
17	0	0
16	0	1
15	0	2

Key part 1

	Min	Max
14	0	0
13	0	0
12	0	1
11	0	1
10	0	3

Key part 2

	Min	Max
9	0	0
8	0	0
7	0	1
6	0	1

Key part 3

	Min	Max
5	0	0
4	1	1
3	2	2
2	3	3

Fig. 4. Minimum and maximum indices

In other words, these tables represent for a desired score which minimal and maximal cell indices in the given column can be used.

For our running example, the corresponding tables are given in Fig. 4. For instance, if a score of 12 is desired from key part 1 onward, only the candidate key parts in row 0 and 1 will match.

Altogether, the preparation of these tables enables a fast pruning of impossible paths, and speeds up the selection of proper candidates.

3.4 The SKEA algorithm

The full SKEA algorithm can then be presented as follows. Given the number of key parts m and a sorted score table $S[i][k]$ for all key part candidates k for key part number i, we first use Alg. 3.1 to precompute the tables containing the cumulated minima, maxima and corresponding index selection limits. These tables are then used for efficient pruning in a depth-first search identifying all possible score paths conforming to a cumulated score equal to $S_{\max}, S_{\max}-1, \ldots$ and so forth, see Alg. 3.2, where line 2 can be efficiently parallelized.

Algorithm 3.1 Generating score bounds for SKEA

Input: Number m of key parts

Input: Score table $S[i][k]$ holding sorted score of candidate k for key part i $(0 \leq i < m)$

Output: $cMin[i]$, the global cumulative minima per key part index: For $0 \leq i < m$, $cMin[i] = \min_k \sum_{j=i}^{m-1} S[j][k]$, $cMin[m] = 0$.

Output: $cMax[i]$, the global cumulative maxima per key part index: For $0 \leq i < m$, $cMax[i] = \max_k \sum_{j=i}^{m-1} S[j][k]$, $cMax[m] = 0$.

Output: $iMin[i][s]$, the minimum index for attaining a certain minimum score s starting from key part i: $iMin[i][s] = \arg\min_k \left(S[i][k] + \mathrm{cMin}[i+1]\right) \geq s$.

Output: $iMax[i][s]$, the maximum index for attaining a certain minimum score s starting from key part i: $iMax[i][s] = \arg\max_k \left(S[i][k] + \mathrm{cMax}[i+1]\right) \leq s$.

Algorithm 3.2 Scored-based key enumeration algorithm (SKEA)

Input: Number m of key parts

Input: Score table $S[i][k]$ holding sorted score of candidate k for key part i $(0 \leq i < m)$

Output: List $\mathcal{L}$ of complete key candidates

```
$cMin, cMax, iMin, iMax \leftarrow$ BOUNDS(S) (using Alg. 3.1)
for $s = cMax[0]$ downto $cMin[0]$ do
    $i \leftarrow 0$
    $k[0] \leftarrow iMin[0][s]$
    $cs \leftarrow s$
    while $i \geq 0$ do
        while $i < m - 1$ do
            $cs \leftarrow cs - S[i][k[i]]$
            $i \leftarrow i + 1$
            $k[i] \leftarrow iMin[i][cs]$
        end while
        add complete key candidate $k$ to $\mathcal{L}$
        while $i \geq 0$ and $k[i] \geq iMax[i][cs]$ do
            $i \leftarrow i - 1$
            if $i \geq 0$ then
                $cs \leftarrow cs + S[i][k[i]]$
            end if
        end while
        if $i \geq 0$ then
            $k[i] \leftarrow k[i] + 1$
        end if
    end while
end for
```

4 Comparison methodology and setting

The primary goal of the comparison is to understand which enumeration algorithm finds the full key faster on a typical workstation.

4.1 Methodology

We perform comparison between OKEA, PKEA, and SKEA in an experimental setting. We run the algorithm implementations on a sample of lists obtained from CPA of a SW AES implementation. We register the running time for finding the correct full key. From the collected running times we estimate success rate [12] of the full key recovery as a function of the running time and the number of traces used to obtain the lists. This is our main comparison metric.

To make the comparison feasible, we limit the maximum running time and maximum memory available to the algorithm implementations. The maximum running time is set to 10 days on a single CPU core. The maximum memory per core is set to 12 GB (the limitation imposed by the configuration of the computing cluster we have at hand).

We note that the imposed limits introduce right censoring to the data we measure. Therefore, computing obvious statistics such as average time to find the key is not straightforward. This is why we choose to use success rate for comparison. Additionally, we analyze and compare the histograms of the obtained data.

In addition to the execution time, we also register and analyze other parameters: the number of key candidates tried and the amount of memory required. This lets us better understand the behavior of the algorithms.

At this point we compare only the enumeration running time. Full key verification in the experiment is done by comparison to the known correct full key. Full key verification for the real-life unknown key setting is addressed in Section 6.

4.2 Experimental setting

1. **Obtaining the lists.** We obtain the lists of correlation coefficients from CPA of a SW AES implementation RijndaelFurious [11] running on an 8-bit AVR microcontroller[3]. We acquire 2000 power consumption traces and perform 20 independent attacks with up to 100 traces. Each attack is performed in an incremental way: we obtain the lists for the number of traces varying from 10 to 100. Each attack for a given number of traces results in a set of 16 lists with 256 correlation values each, together with the corresponding key byte candidate values.
2. **Converting the lists.** We convert the correlation values according to the algorithm requirements. For OKEA and PKEA we use the Bayesian extension (see Sect. 2). For SKEA we use conversion to integer scores (see Sect. 3.1).

[3] The target is chosen only for the sake of efficient comparison; real enumeration on this target is hardly needed because very small amount of traces is required for successful CPA.

3. **Algorithm implementations.** For OKEA, we use the open-source implementation [13]. For PKEA and SKEA, we use our own implementations in C. All the three implementations are single-threaded. Through a unified wrapper, an implementation takes a set of 16 per-key lists and returns the running time, the maximum memory used, and the number of key candidates tried in case the correct key is found within the defined time and memory limits. Otherwise, an out-of-time or out-of-memory notification is returned. We build all the three algorithm implementations for x86_64 Linux with `gcc` using full optimization (`-O3`).
4. **Running the algorithms.** We deploy the algorithms to a computing cluster. The cluster consists of 42 IBM NeXtScale nx360 M4 nodes each having 2 Intel Xeon E5-2680 v2 CPUs and 128 GB RAM, and 64 HP ProLiant SL2x170z G6 nodes each having 2 Intel Xeon X5550 CPUs and 24 GB RAM. To ensure sufficient amount of memory available, we run one experiment instance per node. We run enumeration experiments for the number of traces starting from 60 down to 42 in steps of 2 or 3. With more than 60 traces, enumeration is not required as the full key is successfully recovered by combining the topmost key byte candidates from the lists. With less than 42 traces, the algorithms fail to recover the key in less than 10 core-days for all the 20 experimental samples. For OKEA, all the unsuccessful cases are due to out-of-memory error and not due to the 10-day limit.

5 Comparison results

First, we visualize the success rate of the full key recovery versus number of traces for the attack in Figure 5. The figure shows success rate for different time boundaries: 10 days (the maximum we could run), 1 day, and 1 hour. It can be seen that SKEA provides distinctly higher success rate, especially when it is possible to run key enumeration for longer time.

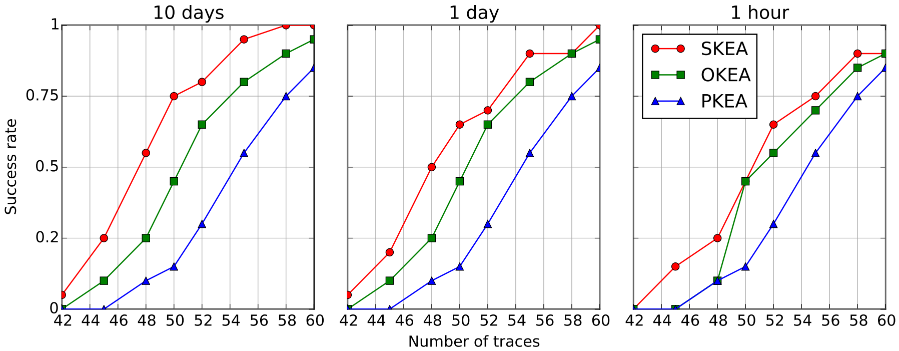


Fig. 5. Success rate of the full key recovery

Next, we compare memory use and number of key tries for SKEA and OKEA. Figure 6 shows the empirical dependency of memory use on the number of enumerated keys. As expected, OKEA memory use grows with the increase in the number of key tires, becoming prohibitive in our setting when the full key is in positions deeper than 2^{32}. SKEA memory use does not depend on enumeration depth. Additional figures are given in Appendix A.

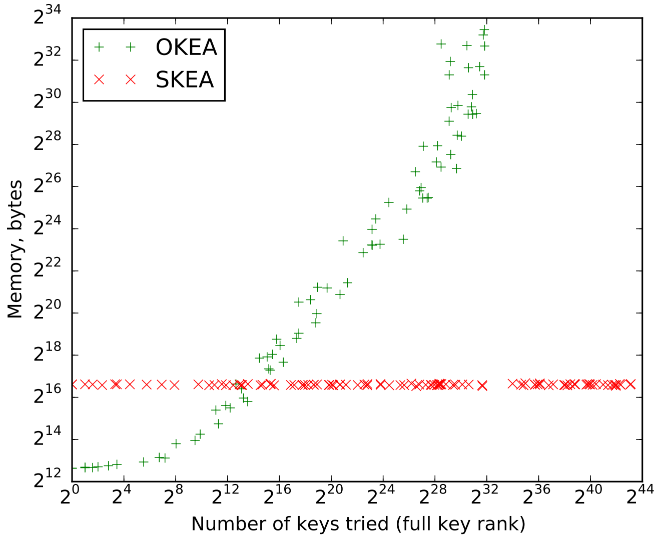


Fig. 6. Memory use versus rank for successful cases

Finally, to understand the relation between the full key rank and the running time, for the experiments where both SKEA and OKEA succeed we calculate SKEA-to-OKEA ratio of the full key rank and SKEA-to-OKEA ratio of the running time. We present the scatter plot of the ratios in Figure 7. The gray lines are boundaries where the corresponding ratios are equal to 1, i.e. the algorithms take the same time resp. make the same number of tries to find the correct key. The quadrants of the plot relative to the $(1, 1)$ point have the following interpretation.

- Bottom right: SKEA takes more time and makes less trials.
- Top right: SKEA takes more time and makes more trials.
- Bottom left: SKEA takes less time and makes less trials.
- Top left: SKEA takes less time and makes more trials.

It can bee seen that even when SKEA makes more trials, in most of the cases it still finds the key faster than OKEA.

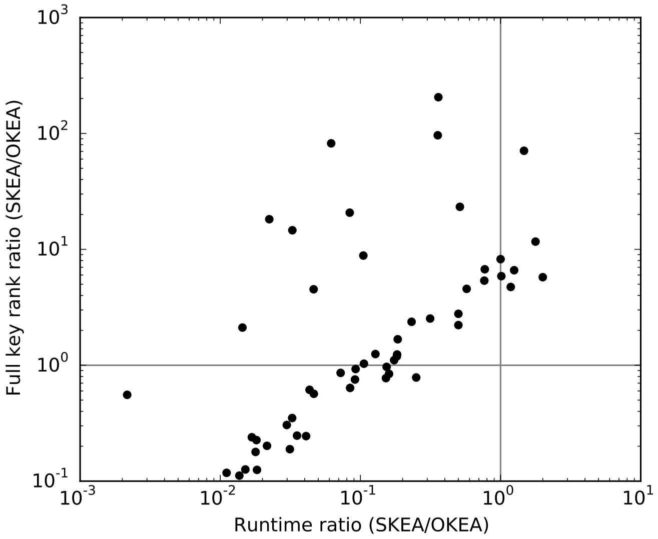


Fig. 7. SKEA-to-OKEA ratio of the full key rank versus SKEA-to-OKEA ratio of the running time

From our empirical comparison we conclude that in practice SKEA appears to be more efficient than OKEA. In particular, it requires less memory, especially when the number of trials (i.e. the full key rank) is high.

6 Combining with key verification

In this section, we discuss how to combine the score-based key enumeration algorithm with an efficient key verification procedure. For concreteness, we focus on AES-128 as the block cipher, and on general-purpose Intel CPUs equipped with the AES-NI instruction set [4, 5].

The setting We consider an architecture in which both SKEA and the key verification process are implemented in software on general-purpose CPUs. The concrete CPU used in our experiments is an Intel Core i5-2400 (Sandy Bridge microarchitecture) with four cores running at 3.1 GHz. For consistency, Turbo Boost was disabled during our measurements.

The key candidates enumerated by SKEA are verified by trial encryption of a small number of known plaintexts and comparing the encryption results with the ciphertexts produced by the correct but unknown key. It is expected that one or two blocks of known plaintext will typically be enough to uniquely identify the key. We note that both the key enumeration (SKEA) and the key verification can be parallelized.

In order to make optimal use of the available resources, it is desirable to have a balanced throughput (measured in keys per second) between both the key enumeration and key verification phases. On our platform, an optimized implementation of SKEA achieves an enumeration throughput of about $2^{25.7}$ keys per second. On the other hand, a straightforward implementation of AES encryption using AES-NI instructions results in a key verification throughput of only about $2^{24.1}$ keys per second. This means that roughly three cores doing key verifications would be necessary to match the performance of the key enumeration on a single core.

The reason for this low AES throughput is that Intel's AES instructions are designed for encrypting large amounts of data with the same key. As a result, the AES round function instructions are heavily pipelined, while the key schedule helper instructions are not: On all microarchitectures from Sandy/Ivy Bridge to Haswell, the `aeskeygenassist` instruction has a latency of 10 cycles with an inverse throughput of 8. Since expanding an AES-128 master key requires 10 applications of `aeskeygenassist` plus some shifts and XORs, this amounts to a latency of around 100 cycles per key.

6.1 Optimized parallel key verification

A solution to overcome this performance bottleneck is based on the observation that one can avoid using `aeskeygenassist` by using the `aesenclast` instruction plus `pshufb` for the S-box step of the key schedule, and then computing the LFSR manually using SSE instructions [2]. By doing this for four keys in parallel, all operations can be carried out on 128-bit XMM registers. In pseudocode, the AES key schedule for four user-supplied keys $k_1, \ldots, k_4$ is then computed as follows:

```
  # register w0: first 32 bits of k1,...,k4
  # ...
  # register w3: last 32 bits of k1,...,k4
  # rk1,...,rk4 contain first round keys corresponding to k1,...,k4
loop from r=1 to 11:
  pshufb     tmp, w3, ROL8_MASK
  aesenclast tmp, ZERO
  aesenc     block1, rk1
  aesenc     block2, rk2
  aesenc     block3, rk3
  aesenc     block4, rk4
  pshufb     tmp, SHIFTROWS_INV
  pxor       tmp, RC[r]
  pxor       w0, tmp
  pxor       w1, w0
  pxor       w2, w1
  pxor       w3, w2
  # ... combine first 32 bits of w0,w1,w2,w3 in rk1
  #     using punpck(l/h)dq and shifts on Sandy/Ivy bridge
  #     or vpunpck(l/h)dq and vpblend on Haswell
  # ... likewise for rk2, rk3, rk4
```

This way, one can keep the instruction pipeline at least partially filled. On Sandy/Ivy Bridge, `aesenc` and `aesenclast` have a latency of 4; on Haswell, the latency is 7. The reciprocal throughput is always 1. This means that the pipeline can be filled completely on Sandy/Ivy Bridge, and to 5/7 on Haswell. This optimized implementation of the AES key schedule results in an improved throughput of about $2^{25.5}$ keys per second on our test platform, yielding a factor of 2.6 speed-up for the key verification process.

With this optimized key verification, we have roughly equal throughput for both steps, which means that one core for key enumeration can be paired to one core for key verification.

6.2 Combined performance measurements

We provide an overview of the performance of key enumeration, verification and their combination on our test platform in Table 1. All measurements we averaged over 200 executions of the algorithm.

Table 1. Performance of combined key enumeration and verification using AES-NI on a Core i5-2400. All numbers are given in keys per second.

	1 core	2 cores	4 cores
(1) SKEA key enumeration	$2^{25.68}$	$2^{26.62}$	$2^{27.59}$
(2) Key verification (naive)	$2^{24.10}$	$2^{25.08}$	$2^{26.09}$
(3) Key verification (optimized)	$2^{25.46}$	$2^{26.44}$	$2^{27.41}$
(1) + (3) combined	$2^{23.87}$	$2^{25.39}$	$2^{26.41}$

Discussion. One can observe that the key enumeration throughput of SKEA scales basically linearly with the number of available cores. The same essentially holds for the key verification, both in the straightforward and the optimized AES-NI implementations. When combining enumeration with verification on a single core, the overall performance becomes limited by the extent to which AES instructions and general purpose instructions needed for SKEA (ALU, memory accesses) can be carried out concurrently while contending for the memory interface. When both enumeration and verification can be assigned to separate cores, however, the combined performance is essentially equal to the minimum of the single-core performances of the two individual tasks. Being able to run two copies of these in the 4-core setup again roughly doubles the achieved throughput. This indicates that off-the-shelf general purpose CPUs can offer an attractive platform for the combined key enumeration and verification.

Acknowledgements. We would like to thank Nicolas Veyrat-Charvillon and François-Xavier Standaert for providing detailed explanations about the Bayesian extension of non-profiled side-channel attacks introduced in [15].

References

1. A. Bogdanov, I. Kizhvatov, K. Manzoor, and M. Witteman. Efficient practical key recovery for side channel attacks. MCrypt Seminar in Cryptography, Les Deux Alpes, 2014. `http://mcrypt.org/pub/kizhvatov_mcrypt.pdf`.
2. A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser. ALE: AES-based lightweight authenticated encryption. In S. Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 447–466. Springer, 2013.
3. M. Dichtl. A new method of black box power analysis and a fast algorithm for optimal key search. *J. Cryptographic Engineering*, 1(4):255–264, 2011. Presented in COSADE'10.
4. S. Gueron. Intel's new AES instructions for enhanced performance and security. In O. Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2009.
5. S. Gueron. *Intel Advanced Encryption Standard (AES) New Instructions Set*. Intel Corporation, 2010.
6. K. Manzoor. Efficient practical key recovery for side-channel attacks. Master's thesis, Aalto University, June 2014. `http://cse.aalto.fi/en/personnel/antti-yla-jaaski/msc-thesis/2014-msc-kamran-manzoor.pdf`.
7. D. P. Martin, J. F. O'Connell, E. Oswald, and M. Stam. Counting keys in parallel after a side channel attack. Cryptology ePrint Archive, Report 2015/689, 2015. `http://eprint.iacr.org/2015/689`.
8. L. Mather, E. Oswald, and C. Whitnall. Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 243–261. Springer, 2014.
9. W. Meier and O. Staffelbach. Analysis of pseudo random sequences generated by cellular automata. In *EUROCRYPT'91*, volume 547 of *LNCS*, pages 186–199. Springer, 1991.
10. J. Pan, J. G. J. van Woudenberg, J. I. den Hartog, and M. F. Witteman. Improving DPA by peak distribution analysis. In *SAC'10*, volume 6544 of *LNCS*, pages 241–261. Springer, 2010.
11. B. Poettering. AVRAES: The AES block cipher on AVR controllers. `http://point-at-infinity.org/avraes/`.
12. F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.
13. N. Veyrat-Charvillon. Key enumeration and key rank estimation software. `http://people.irisa.fr/Nicolas.Veyrat-Charvillon/software.html`.
14. N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F. Standaert. Efficient implementations of a key enumeration algorithm. COSADE'12, session "Work in Progress", 2012.
15. N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F. Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *SAC'12*, volume 7707 of *LNCS*, pages 390–406. Springer, 2013.
16. N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Soft analytical side-channel attacks. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 282–296. Springer, 2014.

A Additional figures

Figure 8 shows empirical distributions (histograms) of the algorithm running time versus the number of traces for the experiment described in Section 4. Bins are spanned along the vertical axis showing the running time, the value of each bin is represented by the color intensity, there 200 bins of width 4320 seconds (1 hour 12 minutes). The last (topmost) bin accumulates cases that ran out of time or (for OKEA) out of memory.

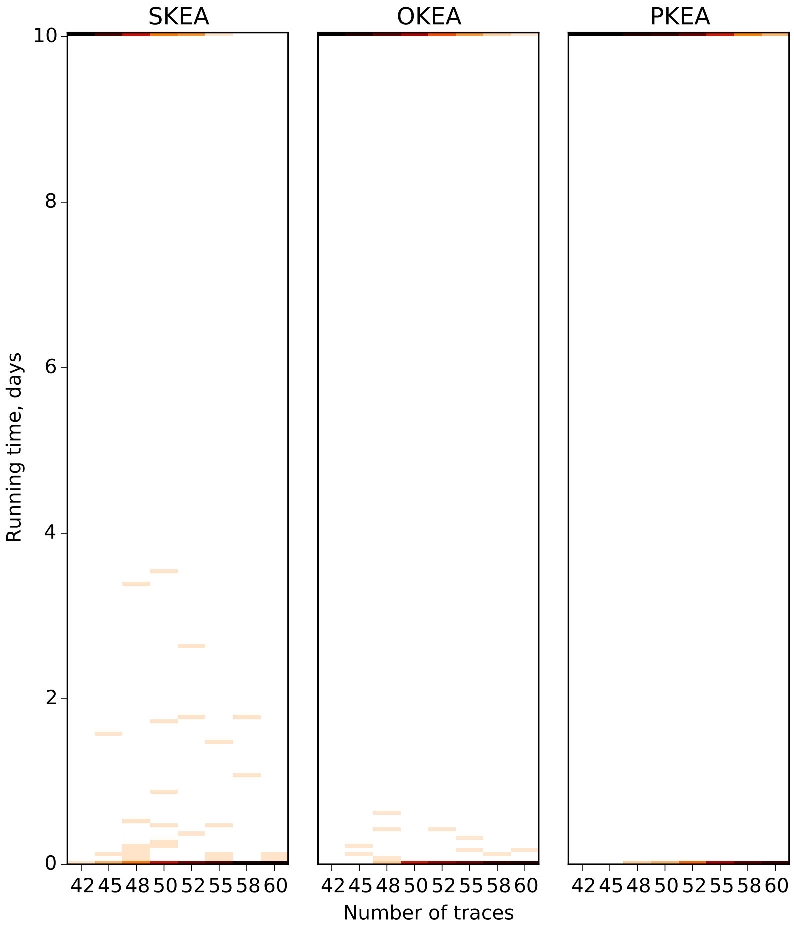


Fig. 8. Empirical distributions of the algorithm running time

Figures 9 and 10 show memory use and the number of full key candidates tried before the correct key is found.

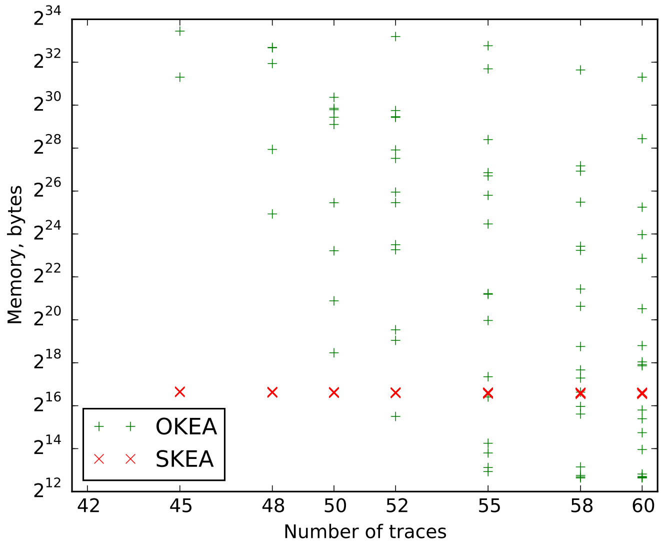


Fig. 9. Memory use for successful cases

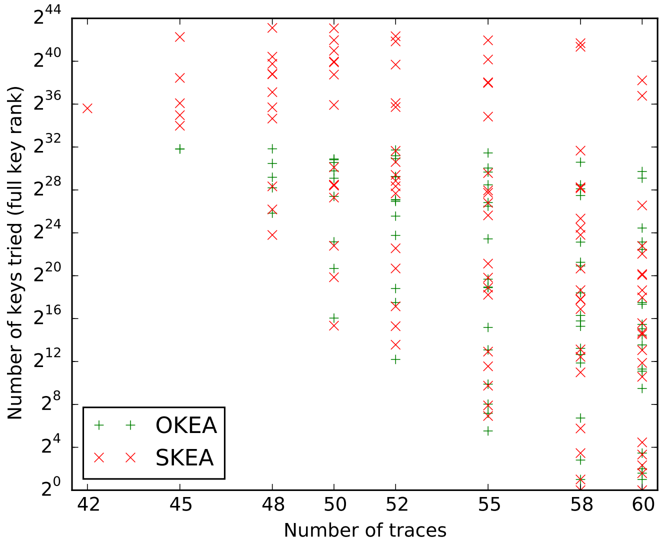


Fig. 10. Number of key tries for successful cases