Fast Leakage Assessment

Oscar Reparaz(✉), Benedikt Gierlichs, and Ingrid Verbauwhede

Department of Electrical Engineering, imec-COSIC, KU Leuven,
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
{oscar.reparaz,benedikt.gierlichs,ingrid.verbauwhede}@esat.kuleuven.be

Abstract. We describe a fast technique for performing the computationally heavy part of leakage assessment, in any statistical moment (or other property) of the leakage samples distributions. The proposed technique outperforms by orders of magnitude the approach presented at CHES 2015 by Schneider and Moradi. We can carry out evaluations that before took 90 CPU-days in 4 CPU-*hours* (about a 500-fold speed-up). As a bonus, we can work with exact arithmetic, we can apply kernel-based density estimation methods, we can employ arbitrary pre-processing functions such as absolute value to power traces, and we can perform information-theoretic leakage assessment. Our trick is simple and elegant, and lends itself to an easy and compact implementation. We fit a prototype implementation in about 130 lines of C code.

Keywords: Leakage assessment · Efficient computation · Side-channel analysis · Countermeasure

1 Introduction

Implementations of cryptographic protocols and algorithms often need to be protected against side-channel attacks. This is true for devices all along the range from tiny embedded compute platforms, where an adversary is able to perform local attacks (power [KJJ99], EM [QS01,GMO01], etc.), to cloud infrastructure, where an attacker is able to perform remote attacks (timing attacks [BB03], cache attacks [Per05], etc.). There is a large variety of countermeasures, some are ad-hoc, others are supported by theory, some protect against specific attacks, others protect against families of attacks, etc. However, most countermeasures have in common that it is not easy to implement them properly, and thus the effectiveness of their implementation needs to be carefully validated. This is done by physical testing. The most common, classical approach is to apply relevant attacks and assess the effort that is required to break the implementation. An advantage of this approach is that one gets a good view on the security level provided by the implementation. A disadvantage is that the approach can be extensive, time consuming and costly. Indeed, an attack may comprise many steps (sample preparation, data acquisition, pre-processing, analysis, post-processing, key enumeration) and for each step there are many possible techniques, and there are many relevant attacks.

W. Fischer and N. Homma (Eds.): CHES 2017, LNCS 10529, pp. 387–399, 2017.
DOI: 10.1007/978-3-319-66787-4_19

Leakage assessment is a fundamentally different approach. It was introduced by Coron, Naccache and Kocher [CKN00,CNK04] after the publication of Differential Power Analysis [KJJ99] as a procedure to assess side-channel information leakage. In brief, leakage assessment techniques allow to assess whether a device leaks information that might be exploitable by side-channel attacks. The approach gained momentum in security evaluations of countermeasures against side-channel attacks in academia [BGN+14,SM15,DCE16] after it resurfaced in publications by Cryptography Research Inc. [GJJR11,CDG+13]. Leakage assessments (especially non-specific ones, see Sect. 2) are easy to carry out and can be very sensitive.

Previous Work. Schneider and Moradi present formulae for leakage assessment at any order traversing the dataset only once [SM15]. They base their approach on the work of Pébay [P08]. The works of Durvaux and Standaert [DS16] and Ding et al. [DCE16] are orthogonal to this paper, and can benefit from the techniques we describe.

Our Contribution. We present a computational trick that serves to accelerate the computationally heavy part of leakage assessment, in any statistical moment or other property of the leakage samples distributions, by orders of magnitude compared to [SM15].

2 Leakage Assessment

There is essentially only one approach to leakage assessment: it is Test Vector Leakage Assessment (TVLA) by Cryptography Research (CRI).

The main idea is to check whether two well chosen data inputs (the test vectors) lead to distinguishable side-channel information. One says that the device leaks if the side-channel information is distinguishable. The test should be repeated with a few pairs of input data to increase confidence. If the side-channel information is not distinguishable one has learned little, in particular one must not conclude that the device does not leak. Even when the device is deemed leaking, one does not necessarily learn that key-extraction is easy.

In the remainder we focus on (local) power analysis attacks, where the side-channel information is typically referred to as power traces or curves. It is straightforward to adapt the technique to other sources of information leakage, e.g. binary cache hit/miss.

Applying TVLA one chooses two inputs, A and B. One obtains measurement traces from the processing of A or B. Practice has shown that tests are very sensitive and pick up all kinds of systematic effects, therefore the measurement process should be as randomized as possible. For example, measurements of groups A and B should be randomly interleaved.

One ends up with two sets of measurements and computes a suitable statistic or metric to determine if the samples in the two sets come from the same distribution or not [GJJR11,CDG+13,MOBW13]. For instance, using Welch's

two-tailed t-test with confidence level 99.999% one determines that the device leaks if the threshold ±4.5 is exceeded.

The TVLA proposals by CRI distinguish specific and non-specific tests. In specific tests one checks for leakage of one or a few chosen intermediate values. One chooses the inputs A and B such that a "substantial" difference occurs only for the chosen intermediate values while all other intermediate values appear to be similar or random. This typically requires to know the key used by the implementation. In non-specific tests one checks for leakage of any intermediate value. Due to fundamental properties of cryptographic algorithms it often suffices to choose two different inputs A and B.

To enhance the power of the assessment and to increase coverage of special cases that are hard to anticipate, CRI proposed fixed versus random testing. One input is fixed at the chosen value, the other input takes uniformly distributed random values (in specific tests the domain can be restricted).

Durvaux and Standaert [DS16] argue that fixed versus fixed testing with two different well chosen inputs can lead to faster leakage detection.

Anyhow, what we propose in the next section is orthogonal to this and hence may serve to improve all flavors of leakage assessment.

3 Fast Leakage Assessment

The focus of this paper is on how one uses the measurement samples to compute the test statistic. A simple approach could be to first obtain all measurements and store them for instance on a hard disk. Then one reads the measurements and uses chosen algorithms to compute the required moments of the distributions and finally the statistic. A different approach could be to interleave data acquisition and computation of statistical moments. There are different algorithms with different properties: some algorithms require multiple passes through the data set, some algorithms are single pass; some algorithms are numerically more stable than others. In general, one wants to use algorithms that are efficient and numerically stable. Typically one ends up using so-called update algorithms. For each new sample, these algorithms update a number of intermediate results that allow to compute the final value without having to pass through the data set again. See [SM15] for a discussion.

Key Idea. The first key observation for our work is that all solutions and algorithms discussed in the side-channel analysis-related literature directly compute moments or statistics from samples. The second key observation is that, in order to compute a distribution's mean, one only needs a sample distribution histogram, and not the whole sample set. The same holds for variance, and actually for any statistical moment or other property of the sampled distribution.

Our main idea is hence a divide and conquer step: we first compute the histogram describing the sample distribution for each class, for each time sample. Only this step requires access to the traces. Note that this step is typically performed quite fast. It boils down to read/write memory accesses and counter

increments by one. Then, using only the histograms, we compute all necessary statistical moments and the t-statistic, or other properties and metrics.

Notation. We assume that an evaluator takes N side-channel leakage traces $t_i[n]$. The time index n ranges from 1 to the trace length L. The trace index i ranges from 1 to the number of traces N. Each time sample within a trace is an integral value from the set $\{0, 1, \ldots, 2^Q - 1\} = [0, 2^Q) \cap \mathbb{Z}$, where Q is the number of quantization bits used to sample the side-channel signal. (In typical oscilloscopes, $Q = 8$ bits.) The array $c[i]$ stores the class index corresponding to the i-th trace. We assume there are just two classes $c[i] \in \{0, 1\}$, $\forall i$ (fixed and random, for instance).

Procedure. The procedure works as follows:

Step 1. Initialize two families of histograms $H_0[n]$ and $H_1[n]$. Each family is actually a $L \times 2^Q$ matrix of counters. Row n stores the histogram for trace distribution at time sample n.

Step 2. For each trace $t_i[n]$, $n = 1, \ldots, L$ belonging to class $c[i] \in \{0, 1\}$, update the corresponding histograms as $H_{c[i]}[n][t_i[n]] \leftarrow H_{c[i]}[n][t_i[n]] + 1$.

Step 3. From the two histogram families H_0 and H_1, compute the necessary moments and the t-statistic value.

Why this Works. This procedure works since histogram families H_i carry all the information about the (estimated) class-conditional distributions. From those histograms, it becomes easy to compute means or any other statistical moment or property of the distributions. For example, we can compute the sample mean $m_0[n]$ at time sample n in step 3 from H_0 as:

$$m_0[n] = \frac{1}{\sum_{i=0}^{2^Q-1} H_0[n][i]} \sum_{i=0}^{2^Q-1} i \cdot H_0[n][i]. \tag{1}$$

In a similar way, we can compute all necessary statistical moments to calculate the t-statistic value at any order only from the histograms H_i.

4 Implementation

We wrote a C99 shared library named `libfastld`, with no external dependencies except `math.h`, implementing the previous methodology. Our approach is simple: it fits in around 130 lines of source code. Our interface is simple as well: the implementation provides a function

```
void add_curve(fastld_t *ctx, uint8_t *curve, uint8_t class);
```

that processes one `curve` array and updates the corresponding histograms in the state `ctx` according to the trace `class`. This curve can be discarded after calling `add_curve`. Afterwards, the state is processed in step 3 to yield a t-statistic

curve. Our current implementation first pre-processes histograms at arbitrary order (by first centering and then exponentiating each histogram) and then uses the Welford method to estimate the variance [Knu81, Sect. 4.2.2.A]. Then, from variances and means the t-statistic is constructed.

We decided to use 32-bit unsigned integers for matrix counters. This means we can iterate step 2 over 2^{32} (≈ 4 billion) traces without problems. This seemed to us like a comfortable maximum number of traces for our current evaluations. We could have used 16-bit counters for better performance, but this requires more care not to overflow (theoretically possible after $2^{16} \approx 65\,\text{k}$ measurements). In our case, measurements are quantized with $Q = 8$ bits.

Which Algorithm to Use for Variance Computation? For the moments computation step, we are not forced to use single-pass algorithms. This step is performed based on a histogram, and not on the whole trace dataset. Therefore, multiple-pass algorithms are cheap to compute, can provide very good numerical stability and can lend themselves to an easy implementation [TFC83].

(Recall that the distribution estimation phase, step 1, is single-pass. Thus, once we acquire a trace and update the corresponding histogram, we can throw away the trace.)

5 Performance Analysis

5.1 Analytical

Separation of Tasks. We effectively decouple two tasks:

1. estimating measurement distributions, and
2. computing distribution parameters (statistical moments).

The first task is performed in step 2. This step produces a compact representation of measurement distributions (namely, two sets of histograms). The running time of this step depends on the number of traces, and only this step. The computational effort in step 2 is just one counter increment per time sample per trace. This is the key advantage of this method: the computational work per trace is minimal.

The resulting histograms are used in step 3 to perform the second task. Previous approaches, such as Schneier and Moradi [SM15] perform both tasks at the same time. We will see that performing the moment estimation from the histogram information brings advantages.

5.2 Empirical

First Dataset. Our dataset comprises $N = 10^6$ traces of $L = 3000$ samples long. We assume traces are provided one-at-a-time. (If traces are made transposed, this may accelerate the process, but we are interested in on-the-fly algorithms.) Our dataset is synthetic: the distribution of each time sample is uniform at random in $\{0, \ldots, 255\}$. We compile the implementation from Sect. 4 with `gcc` version 4.9.2 and optimization flags `-O3`. Our benchmark platform is a Core i5 desktop workstation running at 3.3 GHz.

Running Times. The update of histograms (Step 2 in Sect. 3) takes 9.8 s to process the $N = 10^6$ traces. This makes roughly a trace processing bandwidth of 305 MB/s. After step 2, we compute the first 10 statistical moments (Step 3 in Sect. 3) in 0.8 s. Note that the distribution of this synthetic dataset is the worst possible for cache efficiency. Thus, these figures can be taken as worst-case. Traces coming from an actual device will likely follow a distribution more amenable to cache accesses in step 2.

Memory Requirements. For traces $L = 3000$ time samples long the size required to hold the two histogram families H_0 and H_1 is about 6 MB, which just fits into the L2 cache of modern processors.

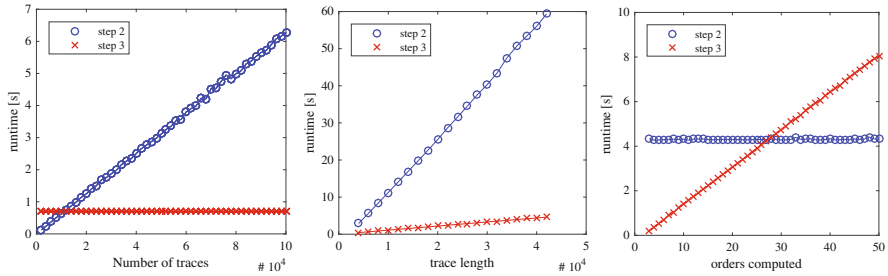

Fig. 1. Left: running times for step 2 and step 3, as the number of traces grows, for $L = 5000$ and five statistical moments computed. Center: running times as the trace length grows, for $N = 100\ 000$ traces and five statistical moments computed. Right: running times as the number of statistical moments computed grows, for $N = 100\ 000$ and $L = 5000$.

5.3 Scaling

Scaling of Running Times. The method proposed in Sect. 3 scales well in several directions. It is easy to see that the running time of step 2 is linear in the number of traces, and the running time of step 3 is constant. In Fig. 1, left, we plot the empirical running time of step 2 and 3 as the number of traces grows. Traces are $L = 5000$ time samples long and step 3 computes the first five statistical moments. We can see that indeed the dependency of the running time of step 2 is linear, and the time spent on step 3 is independent of the number of traces. In Fig. 1, center, we plot running times as a function of the trace length, for a fixed number of traces ($N = 100\ 000$) and five statistical moments. We can see that the running times of both steps depend linearly on the trace length. Finally in Fig. 1, right, we plot the running times as a function of the number of statistical moments computed. Step 2 runs in time independent of the number of computed moments, and the running time for step 3 is linear in the number of computed moments. In this case, we fix the number of traces $N = 100\ 000$ and the trace length $L = 5000$.

Memory Scaling. It is easy to see that memory requirements scale linearly with the trace length L. Interestingly, they are constant with the number of traces N. The number of computed statistical moments incurs little influence on the memory requirements as well. There is an exponential dependency on Q, albeit for a typical oscilloscope we may safely assume $Q = 8$. The number of bytes required to store the histogram families is obviously $2 \times L \times 2^Q \times W$ if we use W-byte counters.

6 Discussion

6.1 Comparison with Other Approaches

As a rough benchmark, the work of Schneider and Moradi requires around 9 h to compute up to the fifth order from a dataset of 100 million traces of 3 000 time samples using 24 cores [SM15, Sect. 4.3]. This makes around 7.8×10^{-3} s per trace per core, which is roughly 800 times slower than our approach. We note that this is a very crude benchmark: the benchmarking platforms are substantially different and many other variables are as well different. Nevertheless, it serves as evidence that the approach presented in this paper outperforms previous work by several orders of magnitude.

We also implemented the update formulae of Schneider and Moradi and benchmarked with the same dataset. The results can be found in Appendix A.

Table 1. Scaling. A cell marked with "—" means "same content as the cell above".

Method	Ops. per trace	Time per trace $L = 3000$	Finalization step $L = 3000$
[SM15], orders 1–5	$14L$ RW + $66L$ MUL + L DIV + $31L$ ADD	7.8×10^{-3} s	0 s (virtually free)
This paper, orders 1–5	$2L$ RW + L ADD	1.45×10^{-5} s	0.3 s
This paper, orders 1–10	—	—	0.8 s
This paper, orders 1–50	—	—	4.6 s
This paper, orders 1–100	—	—	9.4 s

Operation Count Scaling. Here we describe how the operation count scales for both methods. The results are condensed in Table 1. We should note that operations for the [SM15] method are performed on double-precision floating point arithmetic, while for our method are performed using 32-bit integers. The operation count of the [SM15] method was a back-of-the-envelope calculation based on the update formulas in [SM15, Sect. A.1].

If restricted to computing the first five statistical moments, the [SM15] method requires the following operation count per trace per timesample: about

14 double-precision floating-point read-write operations, about 66 floating-point multiplications, 1 floating-point division and about 31 floating-point additions. The number of operations in this case highly depends on the number of statistical moments one would like to compute.

In contrast, the method proposed in this paper requires just two 32-bit memory read-write operations and one integer addition for step 2, per time sample per trace. The computation per trace is constant, independent of the statistical moments one is interested to compute. Our method requires a finalization step that takes less than 1 s for $L = 3000$ traces, and does not depend on N and thus can be amortized. This means that our method outperforms the [SM15] method for sufficiently many traces.

Memory Scaling. For both methods, memory requirements are not usually the bottleneck. Both methods require linearly more memory as the number of time samples L grows. The [SM15] method requires memory linear in the number of statistical moments computed, and our method requires memory exponential in the quantization steps of the oscilloscope, often $Q = 8$.

6.2 Parallelization

The algorithm described in Sect. 3 is embarrassingly parallel. The work of step 2 can be split across several processors trivially. Step 3 can be as well distributed: each processor gets a different time-slice. However, we believe parallelization will be applied seldom. In our case, our implementation is single-threaded because multi-threading was not deemed necessary. Step 2 is no longer the most consuming step in our evaluation chain.

6.3 Bonuses

Bonus: Exact Arithmetic. In the extreme case, one can perform the whole computation of step 3 in rational arithmetic (that is, working in $\mathbb{Q}$). Thus, we have the ability to compute the *exact* value for the (square of the) t-statistic, eliminating any round-off or numeric stability issue. Since we work with exact values, the choice of estimator to compute statistical moments is superfluous, since they will all yield the intended exact result.

The penalty in computational effort is very low. There is no impact on the running time of the dominating step 2, and only slight on step 3. We have implemented step 3 in rational arithmetic using the GNU's Multiple Precision Arithmetic library (GMP) with `mpq_t` rational integer types. (This requires obviously linking against GMP. This is only required for using exact arithmetic.) The final square root floating point computation in the denominator is performed with at least 128-bit precision, but this is not really necessary. We can very well compare t^2 against $(4.5)^2$ to fail or pass a device.

We repeated the experiment with the dataset from Sect. 4. Step 3 now takes around 81 s. This is a one-time effort. We believe this is a useful feature, especially for evaluation labs where extra assurance is desirable. When using exact

arithmetic one does not have to worry about the choice of algorithm for variance, numeric stability or errors.

Bonus: Kernel-Based Estimation. It is well-known that kernel-based density estimation methods may lead to more accurate estimations, especially when the sample size is scarce (few traces). We note that kernel-based estimations are very easy to plug in our method. One can apply the kernels directly to the histograms (output of step 2), instead to each trace individually. Then, the evaluator only has to modify accordingly step 3. This fact can be helpful if the evaluator wants to experiment with a family of kernels (say, different kernel bandwidths). She can experiment with different parameters *a posteriori*, once traces have been collected and thrown away in step 2.

Bonus: Arbitrary Pre-processing Function. Sometimes, one is interested in using a different pre-processing function other than centering and exponentiating. In some noisy cases, it is advisable to use for example absolute difference [BGRV15] rather than the theoretically optimum centered product [PRB09] (assuming HW leakage behavior and Gaussian noise).

The evaluator can apply an arbitrary pre-processing function of his choice after step 2 and before step 3. For example, the evaluator can inject the absolute value pre-processing function. This is not trivial to do in the approach of Schneier and Moradi. (One could develop the Taylor expansion for $|x|$ to approximate with a polynomial, but this incurs an error due to truncation.)

Bonus: Information-Theoretic Leakage Detection. From the histograms, it is also possible to perform an information-theory based leakage detection test, using for instance a two-sample Kolmogorov–Smirnov test (this tool was previously used for instance by Whitnall and Oswald [WOM11]) or mutual information [MOBW13]. We note that for mutual information it is possible to apply a kernel density estimation method *after* the histograms H_0 and H_1 have been estimated (due to linearity), resulting in a fast estimation.

Bonus: Cropping Detection. It is important to detect if there is cropping while acquiring power traces when performing a leakage detection test. Typical oscilloscopes output a saturated value when sampling a value out of range. When performing leakage detection tests with cropped (saturated) values, the obtained t-value is unreliable and should be discarded. (Normally, with saturated samples the t-statistic grows very much. It is easy to explain this: saturated samples carry a small variance, shrinking the denominator and thus the t-value grows.) One should ideally check that there is no cropping while measuring. However, in the case that such check is missing, our method after step 2 allows to detect if any of the traces were cropped (maybe the evaluator did a mistake, or some event (mis)happened), *after* trace collection. It is however impossible to recover from this mistake; that is, it is impossible to "remove" cropped traces once the histograms of step 2 have been filled, meaning that traces must be acquired again. Nevertheless, we think this can bring extra assurance whether to trust

the evaluation or not. This can be implemented in step 3 as follows: verify that the histogram does not take the extremal values (in our case, 0 and $2^Q - 1$). We found this feature useful in our own experience.

6.4 DPA Attacks

In principle one could port the same principles from Sect. 3 to plain DPA attacks. However, the gain is not so strong, since one should keep in general one histogram per plaintext value, instead of just two. Naively the memory requirements are significantly higher. Of course, if the evaluator can choose texts this can be significantly lowered.

6.5 Deployment

The technique presented in this paper is mature and was used in several evaluations performed in the last three years in our lab. Among others, it was used in the evaluation of recently proposed higher-order masking schemes such as [BGN+14, CBRN14, CBR+15, CRB+16] and other publications [CN16, CBG+17].

7 Conclusion

In this paper we presented a simple methodology to significantly alleviate the computation effort required to perform a leakage assessment. Our method is extremely simple and compact to implement (about 130 lines of C code), but has a significant impact on the running time of a leakage assessment. We can lower the running time of leakage assessment evaluations by several orders of magnitude .

Acknowledgments. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the Flemish Government, FWO G.00130.13N, FWO G.0876.14N and Thresholds G0842.13; by the Hercules Foundation AKUL/11/19; through the Horizon 2020 research and innovation programme under grant agreement 644052 HECTOR and Cathedral ERC Advanced Grant 695305. Benedikt Gierlichs is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

A Benchmark of Schneider and Moradi

Here we repeat the analysis from Sect. 5.3 for the method of Schneider and Moradi. We evaluate both methods in the same machine, with the same toolchain and with software written by the same person. Nevertheless, the purpose of this section is only exploratory, and it is possible that a more optimized implementation of either method yields better performance.

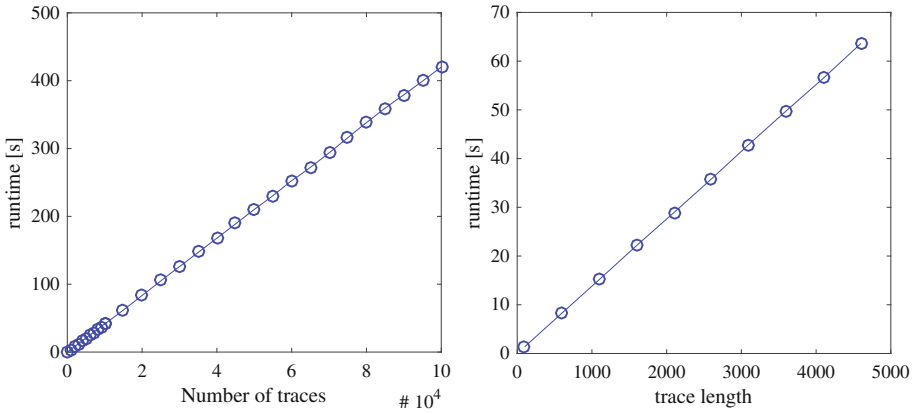

Fig. 2. Method of Schneider and Moradi. Left: running times as the number of traces grows, for $L = 3000$ and five statistical moments computed. Right: running times as the trace length grows, for $N = 100\,000$ traces and five statistical moments computed.

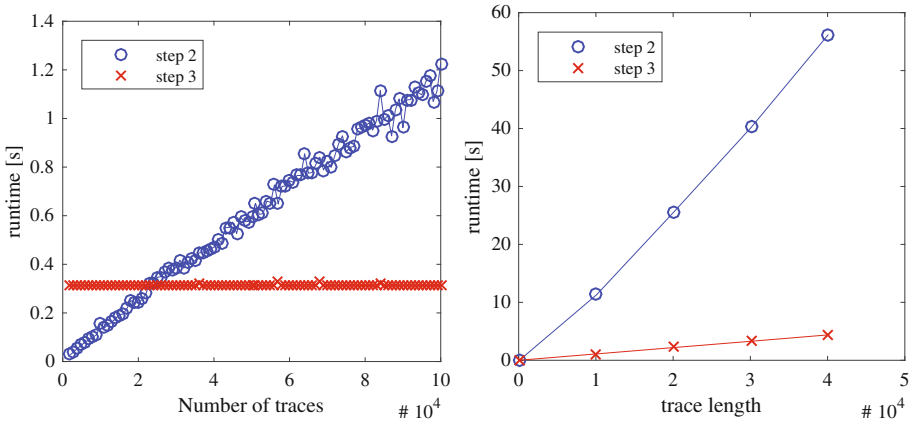

Fig. 3. Method presented in this paper. Left: running times as the number of traces grows, for $L = 3000$ and five statistical moments computed. Right: running times as the trace length grows, for $N = 100\,000$ traces and five statistical moments computed. Note that the number of traces in the right picture is an order of magnitude larger than that of Fig. 2.

We compute t-statistics up to the fifth order. In Fig. 2 we plot the evolution of running times for the method of Schneider and Moradi. For completeness, we repeat in Fig. 3 the performance of the method presented in this paper for parameters that match Fig. 2.

The difference in running time between methods is expected to grow substantially as the statistical order increases. However, estimations of higher-order statistical moments are very sensitive to noise and thus are rarely done in today's evaluations.

References

[BB03] Brumley, D., Boneh, D.: Remote timing attacks are practical. In: Proceedings of the 12th USENIX Security Symposium. USENIX Association (2003)

[BGN+14] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_18

[BGRV15] Balasch, J., Gierlichs, B., Reparaz, O., Verbauwhede, I.: DPA, bitslicing and masking at 1 GHz. In: Güneysu and Handschuh [GH15], pp. 599–619

[CBG+17] De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? In: Guilley, S. (ed.) COSADE 2017. LNCS, vol. 10348, pp. 1–18. Springer, Cham (2017)

[CBR+15] Cnudde, T., Bilgin, B., Reparaz, O., Nikov, V., Nikova, S.: Higher-order threshold implementation of the AES S-Box. In: Homma, N., Medwed, M. (eds.) CARDIS 2015. LNCS, vol. 9514, pp. 259–272. Springer, Cham (2016). doi:10.1007/978-3-319-31271-2_16

[CBRN14] De Cnudde, T., Bilgin, B., Reparaz, O., Nikova, S.: Higher-order glitch resistant implementation of the PRESENT S-Box. In: Ors, B., Preneel, B. (eds.) BalkanCryptSec 2014. LNCS, vol. 9024, pp. 75–93. Springer, Cham (2015). doi:10.1007/978-3-319-21356-9_6

[CDG+13] Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test vector leakage assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference (2013)

[CKN00] Coron, J.-S., Kocher, P., Naccache, D.: Statistics and secret leakage. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 157–173. Springer, Heidelberg (2001). doi:10.1007/3-540-45472-1_12

[CN16] De Cnudde, T., Nikova, S.: More efficient private circuits II through threshold implementations. In: Maurine, P., Tunstall, M. (eds.) International Workshop on Fault Diagnosis and Tolerance in Cryptography 2016, Volume Conference Publishing Service, pp. 114–124, Santa Barbara, CA, USA. IEEE (2016)

[CNK04] Coron, J.-S., Naccache, D., Kocher, P.C.: Statistics and secret leakage. ACM Trans. Embedded Comput. Syst. **3**(3), 492–508 (2004)

[CRB+16] De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with $d+1$ shares in hardware. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 194–212. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53140-2_10

[DCE16] Ding, A.A., Chen, C., Eisenbarth, T.: Simpler, faster, and more robust T-test based leakage detection. In: Standaert, F.-X., Oswald, E. (eds.) COSADE 2016. LNCS, vol. 9689, pp. 163–183. Springer, Cham (2016). doi:10.1007/978-3-319-43283-0_10

[DS16] Durvaux, F., Standaert, F.-X.: From improved leakage detection to the detection of points of interests in leakage traces. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 240–262. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49890-3_10

[GH15] Güneysu, T., Handschuh, H. (eds.): CHES 2015. LNCS, vol. 9293. Springer, Heidelberg (2015)

[GJJR11] Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance validation. In: NIST Non-invasive Attack Testing Workshop (2011)

[GMO01] Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). doi:10.1007/3-540-44709-1_21

[KJJ99] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25

[Knu81] Knuth, D.E.: The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd edn. Addison-Wesley, Boston (1981)

[MOBW13] Mather, L., Oswald, E., Bandenburg, J., Wójcik, M.: Does my device leak information? an *a priori* statistical power analysis of leakage detection tests. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8269, pp. 486–505. Springer, Heidelberg (2013). doi:10.1007/978-3-642-42033-7_25

[P08] Pébay, P.: Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical report SAND2008-6212, Sandia National Laboratory (2008)

[Per05] Percival, C.: Cache missing for fun and profit. In: Proceedings of BSDCan 2005 (2005)

[PRB09] Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)

[QS01] Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). doi:10.1007/3-540-45418-7_17

[SM15] Schneider, T., Moradi, A.: Leakage assessment methodology - a clear roadmap for side-channel evaluations. In: Güneysu and Handschuh [GH15], pp. 495–513

[TFC83] LeVeque, R.J., Chan, T.F., Golub, G.H.: Algorithms for computing the sample variance: analysis and recommendations. Am. Stat. **37**(3), 242–247 (1983)

[WOM11] Whitnall, C., Oswald, E., Mather, L.: An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 234–251. Springer, Heidelberg (2011). doi:10.1007/978-3-642-27257-8_15