Optimization of Pearson correlation coefficient calculation for DPA and comparison of different approaches

Petr Socha, Vojtěch Miškovský, Hana Kubátová, Martin Novotný
Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design
{sochapet,miskovoj,kubatova,novotnym}@fit.cvut.cz

***Abstract*—Differential power analysis (DPA) is one of the most common side channel attacks. To perform this attack we need to calculate a large amount of correlation coefficients. This amount is even higher when attacking FPGAs or ASICs, for higher order attacks and especially for attacking DPA protected devices. This article explains different approaches to the calculation of correlations, describes our implementation of these approaches and presents a detailed comparison considering their performance and their properties for a practical usage.**

***Index Terms*—Differential Power Analysis, DPA, Pearson correlation coefficient, AES, Side Channel Attack**

I. Introduction

Side channel attack (SCA) is a common threat for cryptographic devices. Even modern cryptographic systems based on mathematically secure ciphers like AES can be vulnerable to SCA.

One of the most commonly used SCAs is differential power analysis (DPA) [1], and especially its correlation based variant (CPA) [2]. This kind of attack is based on measuring the power traces of a cryptographic device during multiple encryptions, with random plain texts, and then computing correlations between the measured power consumption traces and a chosen power model based on a plain/cipher text and possible values of a key. There are also other similar attacks based on other quantities (e.g. electromagnetic radiation) [3].

To perform the attack against modern low-power devices, we need to obtain a huge amount of power traces. Also, higher order DPA demands more power traces than the first order one [4], [5]. Afterwards, the correlation calculation can get highly time demanding.

In this paper we compare several methods of Pearson correlation coeficient computation on large amount of data and discuss their properties. In Section II we present works related to this topic. In Section III we introduce the mathematical background for each method and discuss its advantages and disadvantages. Efficient implementations of these methods are proposed in Section IV. A comparison of the performance of the methods for attacking AES cipher is presented in Section V. We conclude our results in Section VI and propose some possible future improvements in Section VII.

II. Related work

In [1] and [6], differential power analysis was introduced as a side channel attack posing a threat e.g. for implementations of DES or AES.

Enhanced correlation variant of differential power analysis, called Correlation Power Analysis (CPA), was introduced in [7] and [8]. Proposed method attacks a byte of the key at a time using correlation coefficients.

Several implementations of differential power analysis attacks have been published (e.g. [9], [10]), however used methods of computing correlation coefficients may suffer from a poor numerical stability [11].

In [12], various aspects of CPA are discussed, some algorithms are suggested and compared, however presented methods may be numerically unstable as well.

A numerically stable approach of computing correlations for side channel attacks is introduced in [13], however no implementation or performance comparison of discussed methods is presented.

To the best of our knowledge, no comparison of hereby presented Pearson correlation coefficient computation methods, that would provide tangible figures and discuss their properties, is available in open literature. In the following sections, we describe the theoretical background of these methods, discuss their advances and limitations and present the experimental work regarding their performance.

III. Correlation

In the process of obtaining a keyguess, a correlation coefficient between each sample and an expected power leakage (hereafter referred to as power model) must be computed. Since we assume a linear dependence between the measured variables [7], Pearson correlation coefficient is used.

The Pearson correlation coefficient between random variables X and Y is defined as follows [14]:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}, \tag{1}$$

with $\mathrm{cov}(X,Y)$ being the covariance between variables X and Y, and σ_X, σ_Y being standard deviations of variables X, Y respectively.

Different approaches to implementing a calculation of correlation coefficient on a statistical sample will be described in following subsections, since the selected method of processing the data has a significant effect on both time and memory performance, as well as on some other properties of the calculation.

A. Two pass approach

Given that $\mathrm{cov}(X, Y) = \mathrm{E}[(X - \mu_X)(Y - \mu_Y)]$ and $\sigma_X^2 = \mathrm{E}[(X - \mu_X)^2]$, with μ_X, μ_Y being means of X, Y respectively [14], we can express the Pearson correlation coefficient in a format more suitable for the processing statistical sample:

$$\rho_{X,Y} = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sqrt{\mathrm{E}[(X - \mu_X)^2]}\sqrt{\mathrm{E}[(Y - \mu_Y)^2]}}. \tag{2}$$

Based on formula (2), a two pass algorithm for the correlation computation can be easily designed. When the correlation is computed on a statistical sample, letter r will be used instead of letter ρ. Assuming we have datasets $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$ with cardinalities n and sample means $\overline{x}$ and $\overline{y}$, we can write:

$$\mathrm{r}_{X,Y} = \frac{\sum\limits_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum\limits_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum\limits_{i=1}^{n}(y_i - \overline{y})^2}}. \tag{3}$$

The first time the data sets are passed, the sample means $\overline{x}$ and $\overline{y}$ are calculated. With the second pass, the variances (thus standard deviations) and the covariance are calculated, allowing to easily compute the final correlation coefficients.

Unless the amount of statistical samples n is very large, this method is numerically stable and it is giving accurate results [11]. However, the two (or more) pass computation is undesirable, especially for large data sets.

For once, the time necessary for the calculation grows with more statistical samples in the set, thus the time performance issue gets significant where more power traces may be needed (on FPGAs, when processing higher order attack, etc.) [12]. Multiple passing algorithms are impractical in this case, since the distributed memory access may dominate the computation time.

Also, the final application asks for the ability to stop the computation, check the results and in the case of need, to continue the computation with more measured samples, allowing e.g. to find out the number of samples nessesary to obtain the valid keyguess.

A new added sample in this case requires pass through all the already processed data once again.

B. Naive one pass approach

When we expand the formula (2) using the linearity of the expectation and the variance [14], we obtain:

$$\rho_{X,Y} = \frac{\mathrm{E}[XY] - \mathrm{E}[X]\mathrm{E}[Y]}{\sqrt{\mathrm{E}[X^2] - \mathrm{E}[X]^2}\sqrt{\mathrm{E}[Y^2] - \mathrm{E}[Y]^2}}, \tag{4}$$

giving us a convenient one pass approach to the computation of the correlation coefficient:

$$\mathrm{r}_{X,Y} = \frac{n\sum\limits_{i=1}^{n} x_i y_i - \sum\limits_{i=1}^{n} x_i \sum\limits_{i=1}^{n} y_i}{\sqrt{n\sum\limits_{i=1}^{n} x_i^2 - (\sum\limits_{i=1}^{n} x_i)^2}\sqrt{n\sum\limits_{i=1}^{n} y_i^2 - (\sum\limits_{i=1}^{n} y_i)^2}}. \tag{5}$$

In a single pass through all the data (statistical samples), the sums $\sum_{i=1}^{n} x_i y_i$, $\sum_{i=1}^{n} x_i$, $\sum_{i=1}^{n} x_i^2$, $\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n} y_i^2$ are computed, allowing to easily calculate the sample means, the variances and the covariance, and finally the correlation coefficients.

This approach promises a better time performance of the implemented calculation, since all the data need to be accessed only once. Also, since the summation of disjunct pools of statistical samples (measured power traces) is an independent operation, this kind of one pass calculation allows for a straightforward parallelization.

Since the critical part of the computation is the summation, the algorithm can be stopped at any time, providing the answer based on processed samples, and then resumed with a little extra effort.

However, this method suffers from a poor numerical stability [11]. Consider the denominator of the formula (5), where the sample variance is computed as a difference of two positive floating point numbers. Due to this subtraction, severe cancellations may occur, leaving the result dominated by a roundoff. A similar situation may occur in the numerator of the formula (5).

C. Incremental one pass approach

A poor numerical stability of this naive one pass algorithm, combined with the necessity to process lot of statistical samples in a reasonable time, demands a better approach. This new approach should preferably maintain both the time complexity of naive one pass algorithm and the numerical stability of the two pass algorithm.

Let's assume that we have datasets $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$ with cardinalities n. Let's assume that $X' = X \cup \{x_{n+1}\}$ and $Y' = Y \cup \{y_{n+1}\}$ are the datasets created by adding some new samples to the sets.

A recurrent formula for the sample mean can be easily obtained and is widely known:

$$\overline{x'} = \overline{x} + \frac{x_{n+1} - \overline{x}}{n+1}. \tag{6}$$

Now, let's assume we have the sample variance defined as $\sigma_X^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$, where n is the cardinality of the dataset. Let $M_{2,X} = \sum_{i=1}^{n}(x_i - \overline{x})^2$, therefore $\sigma_X^2 = \frac{1}{n}M_{2,X}$.

For the dataset X' created by adding a new statistical sample to the set X, we would like to calculate the variance as $\sigma_{X'}^2 = \frac{1}{n+1}M_{2,X'}$. Using the formula (6), a recurrent formula for the sum $M_{2,X'}$ can be directly used [15]:

$$M_{2,X'} = M_{2,X} + (x_{x+1} - \overline{x})(x_{x+1} - \overline{x'}). \tag{7}$$

For the covariance, a similar recurrent formula exists. Assume a dataset consisting of pairs $S = \{(x_i, y_i) | x_i \in X, y_i \in Y\}$, representing pair of random variables X and Y, with the cardinality n and the sample means $\overline{x}$ and $\overline{y}$. The sample covariance is defined as $\operatorname{cov}(X, Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$. Let $C_{2,S} = \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$, then $\operatorname{cov}(X, Y) = \frac{1}{n}C_{2,S}$.

Let's assume that $S' = S \cup \{(x_{n+1}, y_{n+1})\} = \{(x'_i, y'_i)| x'_i \in X', y'_i \in Y'\}$ is a new dataset created by adding a new pair of samples. The covariance of the dataset S' can be then computed as $\operatorname{cov}(X', Y') = \frac{1}{n}C_{2,S'}$, where [16]:

$$C_{2,S'} = C_{2,S} + \frac{n}{n+1}(x_{n+1} - \overline{x})(y_{n+1} - \overline{y}). \tag{8}$$

Substituting the sums in the formula (3) for $C_{2,S}$, $M_{2,X}$ and $M_{2,Y}$, we get the final correlation coefficient:

$$r_{X,Y} = \frac{C_{2,S}}{\sqrt{M_{2,X}}\sqrt{M_{2,Y}}}. \tag{9}$$

Using these recurrent formulas, one can calculate correlation coefficients, even for a large number of samples (measured power traces), assuring numerical stability of the computation [16].

This approach, allowing an online and direct update of all the values every time a new sample is added to the dataset, is well-suited for CPUs, as well as for streaming processors, e.g. graphical processing units. This algorithm can be parallelized just as easily, as the naive one pass algorithm.

IV. Implementation

Attacking AES implementation using the correlation variant of differential power analysis consists of attacking each byte of the key separately.

Each of n measurements of power traces is done with m samples per trace, resulting in a vector of random variables $\mathbf{X} = [X_1, ..., X_m]$, one for each sample.

The plain/cipher text used while measuring the traces, after being processed by an appropriate power model, results in a vector of variables $\mathbf{Y} = [Y_1, ..., Y_{256}]$, one for each key candidate.

To obtain a valid keyguess, the correlation coefficient for each sample per trace and each key candidate must be computed. The result of the algorithm is the correlation matrix $\mathbf{C}$ of size $m \times 256$, where $C_{i,j} = r_{X_i,Y_j}$, computed for each byte of keyguess separately.

The keyguess can be found in the correlation matrix $\mathbf{C}$ simply by searching for the maximal and the minimal correlation coefficient, or by some other statistical method.

A. Two pass approach

As stated before, the two pass algorithm calculates the sample means first. By going through all the data, a vector $\overline{\mathbf{X}} = [\overline{x_1}, ..., \overline{x_m}]$ of m sample means, one for each sample per trace, and a vector $\overline{\mathbf{Y}} = [\overline{y_1}, ..., \overline{y_{256}}]$ of 256 sample means, one for each key candidate, are calculated.

In the second pass, with the precomputed sample means, vectors containing variances $\sigma^{\mathbf{2}}_{\mathbf{X}} = [\sigma^2_{X_1}, ..., \sigma^2_{X_m}]$ and $\sigma^{\mathbf{2}}_{\mathbf{Y}} = [\sigma^2_{Y_1}, ..., \sigma^2_{Y_{256}}]$, as well as a covariance matrix $\mathbf{K}$ of size $m \times 256$, where $K_{i,j} = \operatorname{cov}(X_i, Y_j)$, are easily computed, from which the final correlation matrix $\mathbf{C}$ is easily derived.

Unfortunately, adding new samples into the datasets (adding more power traces) results in running the whole computation from the very beginning, making this implementation unsuitable for many kinds of applications, for example measuring amount of traces necessary to obtain a keyguess. Also, reading all the data twice leads to an expensive distributed memory access.

The whole process is repeated for each byte of the keyguess with the same measured power traces (X), but a different power model, based on the plain/cipher text used (Y).

Our early implementation of the two pass algorithm, written in C++ using Standard Template Library (STL), including the power-model computation, while faster than some universal mathematical software (Matlab, Mathematica) [17], is still unbearably slow for a large amount of traces.

B. Naive one pass approach

In a single pass through the measured data and power model, a vector of sums $\mathbf{S_X} = [\sum_k x_{1_k}, ..., \sum_k x_{m_k}]$ of power traces for each sample and a vector of their powers $\mathbf{S_{X^2}} = [\sum_k x^2_{1_k}, ..., \sum_k x^2_{m_k}]$, as well as vectors of sums of the power model $\mathbf{S_Y} = [\sum_k y_{1_k}, ..., \sum_k y_{256_k}]$ and $\mathbf{S_{Y^2}} = [\sum_k y^2_{1_k}, ..., \sum_k y^2_{256_k}]$ are computed, as well as a matrix of sums $\mathbf{S_{XY}}$ of size $m \times 256$, where $S_{XY_{i,j}} = \sum_k x_{i_k} y_{j_k}$. From these, based on the formula (5), the final correlation matrix $\mathbf{C}$ is easily computed.

This kind of approach offers much better time and memory performance, given an online character of the algorithm. The bottleneck of the whole computation is a multiplication and a summation while creating the matrix $\mathbf{S_{XY}}$. Fine-grain parallelization of this operation results in a very good scalability of this algorithm. The minimum amount of the working memory needed is limited only by a size of vectors and the matrix of sums, mentioned above.

Our implementation, written in C/C++ and using Open Multi-Processing API (OpenMP), gets much faster than our two pass implementation, also thanks to the better memory usage. The final correlation computation was separated from the power model computation, where a rearrangement of the data for a better correlation computation performance also happens. The time performance of the power model computation is well satisfactory on its own, as well as it is scalable.

The properties of the one pass algorithm allow processing a certain amount of samples, giving the answer and continuing the calculation without the need to start from the very beginning, unlike two pass algorithm.

However, as stated earlier, this algorithm can suffer from a poor numerical stability, mostly due to the subtractions.

C. Incremental one pass approach

The incremental one pass algorithm keeps a matrix and vectors of the same values as the two pass algorithm does (unlike naive one pass algorithm, which keeps sums of powers), but rather than computing them in two passes, it uses recurrent update formulas, mentioned earlier.

In a one pass, sample mean vectors $\overline{\mathbf{X}} = [\overline{x_1}, ..., \overline{x_m}]$ and $\overline{\mathbf{Y}} = [\overline{y_1}, ..., \overline{y_{256}}]$ are calculated using the formula (**??**). In the same pass, variance vectors $\mathbf{M_{2,X}} = [M_{2,X_1}, ..., M_{2,X_m}]$ and $\mathbf{M_{2,Y}} = [M_{2,Y_1}, ..., M_{2,Y_{256}}]$ are computed using formulas (6) and (7) and the covariance matrix $\mathbf{K}$, where $K_{i,j} = C_{2,S_{i,j}}$ and $S_{i,j} = \{(x,y)|x \in X_i, y \in Y_j\}$, gets computed using the formula (8). The final matrix $\mathbf{C}$ with Pearson correlation coefficients is then obtained using the formula (9).

While this algorithm is numerically stable [16], it keeps the online character of the naive one pass algorithm and allows optimized distributed memory access and thus a good time performance, as well as a good memory space performance. The bottleneck of the implementation remains in the update of the covariance matrix $\mathbf{K}$, allowing to parallelize easily (e.g. using OpenMP).

The algorithm is also capable of stopping, giving out the results and continuing the computation with more samples, thus allowing to find out the number of traces needed to obtain the valid keyguess, etc.

V. Results

In this section, we will compare the time performance and the scalability of implementations stated earlier. All the time measurements were done on machine equipped with Intel i5 2400 quad core processor at 3.3GHz, 8GB DDR3 667MHz working memory, SSD hard drive and base installation of 64bit Arch Linux. All the implementations were compiled using GNU C Compiler with level 3 optimizations enabled. Wall-clock time is presented and hereafter referred to as running time. Parallel computational threads are hereafter referred to as threads.

A. Two pass approach

Our original two pass implementation, written in C++ using Standard Template Library (STL), combines the computation of correlations and the power model computation. The library containers and a poor memory cache usage, caused by the organization of the computation, lead to a very poor time performance.

Table I presents the running time for a various numbers of power traces and samples per trace, using a single thread.

Table I: Running time of the two pass algorithm for a various numbers of power traces and samples, in seconds

# of traces / # of samples per trace	100	1k	10k	100k
10	0.009	0.127	1.882	73.56
100	0.084	1.361	16.70	794.5
1,000	0.704	11.58	149.6	6 964

As can be seen, for 100,000 power traces and 1,000 samples per trace, the computing time gets unbearably high. This leads to the need for other implementations.

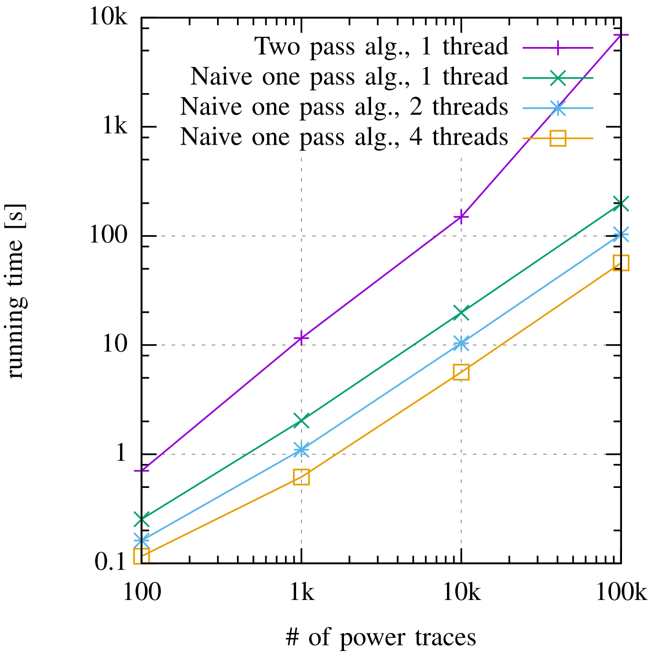

Figure 1: Comparison of the time performance of the two pass and the naive one pass algorithm for a various number of power traces, 1,000 samples per trace, and in a case of the naive one pass algorithm, a various number of threads

B. Naive one pass approach

The optimized naive one pass algorithm, implemented in C language, takes the measured data and the pre-computed power model (see Section V-D), and results with the correlation matrix for each byte of keyguess, and a keyguess based on the max/min correlation coefficient.

Since the naive one pass algorithm is well parallelizable, our implementation uses Open Multi-Processing API (OpenMP) to achieve better time performance.

Tables II and III present the running time for a various number of power traces, samples per trace and number of threads.

As can be seen, the naive one pass algorithm has a good scalability. Also, the computation time grows linearly with the number of traces.

Table II: Running time of the naive one pass algorithm for a various number of power traces and 1,000 samples per trace, in seconds

# of traces / # of threads	100	1k	10k	100k	1M
1	0.253	2.035	19.80	198.7	2006
2	0.162	1.097	10.42	103.7	1049
4	0.116	0.617	5.648	56.47	587.7

Table III: Running time of the naive one pass algorithm for a various number of samples per trace and 100,000 power traces, in seconds

# of samples per trace / # of threads	10	100	1,000
1	2.616	19.53	198.7
2	2.165	10.93	103.8
4	2.271	6.795	56.47

The scalability of the algorithm gets better with a higher number of samples per a trace. This is due to the fine grain parallelism, implemented over the computation of the matrix sized $m \times 256$, where m is the number of samples per trace.

Figure 1 demonstrates the scalability of the naive one pass algorithm. Compared to our original two pass algorithm, our implementation of the naive one pass algorithm gets up to 35× faster when processing 100,000 power traces with 1,000 samples per trace.

C. Incremental one pass approach

Our implementation of the incremental one pass algorithm, also written in C language, has very similar characteristics, as the naive one pass algorithm.

Tables IV and V present the running time of the incremental one pass algorithm for a various amount of power traces, samples per trace and threads.

Table IV: Running time of the incremental one pass algorithm for a various number of power traces and 1,000 samples per trace, in seconds

# of traces / # of threads	100	1k	10k	100k	1M
1	0.292	2.435	23.75	236.7	2383
2	0.184	1.320	12.69	124.9	1267
4	0.132	0.790	7.134	69.52	719.1

Table V: Running time of the incremental one pass algorithm for a various number of samples per trace and 100,000 power traces, in seconds

# of samples per trace / # of threads	10	100	1,000
1	4.074	24.69	236.7
2	3.720	14.43	124.9
4	3.707	10.99	69.52

As can be seen on Figure 2, the incremental algorithm is approximately $1.2\times$ slower, than the naive one pass algorithm.

D. Separate power model computation

Our original two pass implementation computes both power leakage model and the correlation coefficients. For the one pass implementations, we have separated the power model computation from the correlation computation.

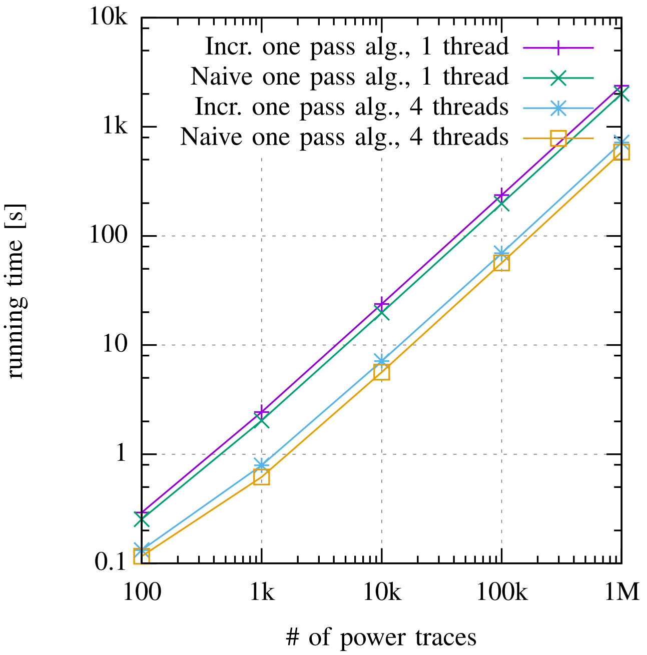

Figure 2: Comparison of the time performance of the naive and the incremental algorithm for a various number of power traces and 1,000 samples per trace, using 1 or 4 threads

The power model computation implementation takes the plain/cipher text, used while measuring, and results with precomputed power model organized for the best correlation computation performance. Its running time depends linearly on the number of power traces and the scalability is quite satisfactory.

Table VI: Running time of the power model precomputation, in seconds

# of traces / # of threads	100	1k	10k	100k	1M
1	0.005	0.052	0.509	5.097	49.80
2	0.002	0.027	0.266	2.665	25.22
4	0.003	0.015	0.141	1.355	13.13

In comparison with correlation coefficients computation, one can neglect the power leakage model computation time as unsignificant, as can be seen in Table VI.

VI. Conclusions

We have compared three algorithms for calculating Pearson correlation coefficients of two large matrices. This calculation is necessary for performing the correlation based DPA. The comparison was based on both mathematical and performance properties.

A simple two pass algorithm proved to be slow, high memory demanding and not well parallelizable. Compared to that, a naive one pass algorithm is much faster, low memory demanding and well paralellizable, but not numerically stable.

Finally, we presented incremental version of the one pass algorithm, which is, unlike the naive one pass algorithm, numerically stable, while being only about 20% slower. This incremental version of the one pass algorithm also preserves other advantages, including low memory demands, satisfactory scalability and an ability to resume the stopped calculation with more added power traces.

While many researches may still use straightforward, two pass algorithm, or numerically unstable naive one pass algorithm, our comparison proves the incremental one pass algorithm to be the best choice for researchers who are mounting DPA attack on any cryptographic device. We have not found any such comparison, providing tangible figures, in open literature.

The incremental one pass implementation helps us to speed up DPA significantly. It also allows for an analysis of an unlimited amount of power traces thanks to its constant memory demands. These properties may be further exploited e.g. for higher order attacks, where more power traces are necessary for obtaining a valid keyguess compared to the first order attack, or for attacking cryptographic devices secured against DPA.

VII. Future work

Since the incremental one pass algorithm, using recurrent formulas stated in this work, is numerically stable and well suited for stream processing, it could be well implemented on Graphical Processing Units (GPUs). We plan to use Open Computing Language (OpenCL) or CUDA and compare all results obtained using hereby presented methods. We expect a better time performance and an expansion of usability of the proposed methods for side channel attacks based on a correlation, especially DPA.

Also, our implementation will be released to public, as a part of DPA software toolkit, allowing anyone for further usage and improvements.

Acknowledgment

This research has been partially supported by the grant GA16-05179S of the Czech Grant Agency, "Fault-Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features" (2016-2018) and CTU project SGS17/017/OHK3/1T/18.

References

[1] P. Kocher, J. Jaffe, and B. Jun, *Differential Power Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, *Power Analysis, What Is Now Possible...* Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 489–502.

[3] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*. Springer, 2001, pp. 200–210.

[4] T. S. Messerges, "Using second-order power analysis to attack dpa resistant software," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 238–251.

[5] J. Waddle and D. Wagner, "Towards efficient second-order power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 1–15.

[6] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards." *Smartcard*, vol. 99, pp. 151–161, 1999.

[7] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[8] T.-H. Le, J. Clédière, C. Canovas, B. Robisson, C. Servière, and J.-L. Lacoume, "A proposition for correlation power analysis enhancement," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 174–186.

[9] K. B. Loğoğlu and T. K. Ateş, "Speeding-up pearson correlation coefficient calculation on graphical processing units," in *Signal Processing and Communications Applications Conference (SIU), 2010 IEEE 18th*. IEEE, 2010, pp. 840–843.

[10] H. Gamaarachchi, R. Ragel, and D. Jayasinghe, "Accelerating correlation power analysis using graphics processing units (gpus)," in *Information and Automation for Sustainability (ICIAfS), 2014 7th International Conference on*. IEEE, 2014, pp. 1–6.

[11] N. J. Higham, *Accuracy and stability of numerical algorithms*. Siam, 2002.

[12] P. Bottinelli and J. W. Bos, "Computational aspects of correlation power analysis," *Journal of Cryptographic Engineering*, pp. 1–15, 2015.

[13] T. Schneider, A. Moradi, and T. Güneysu, "Robust and one-pass parallel computation of correlation-based attacks at arbitrary order," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2016, pp. 199–217.

[14] F. E. Croxton and D. J. Cowden, *Applied general statistics*. Prentice-Hall, 1940.

[15] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Updating formulae and a pairwise algorithm for computing sample variances," in *COMPSTAT 1982 5th Symposium held at Toulouse 1982*. Springer, 1982, pp. 30–41.

[16] P. Pébay, "Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments," *Sandia Report SAND2008-6212, Sandia National Laboratories*, vol. 94, 2008.

[17] V. Miškovský, H. Kubátová, and M. Novotný, "Influence of fault-tolerant design methods on differential power analysis resistance of aes cipher: Methodics and challenges," in *Embedded Computing (MECO), 2016 5th Mediterranean Conference on*. IEEE, 2016, pp. 14–17.