Computational Aspects of Correlation Power Analysis

Paul Bottinelli[1⋆] and Joppe W. Bos[2]

[1] EPFL, Lausanne, Switzerland
[2] NXP Semiconductors, Leuven, Belgium

Abstract. Since the discovery of simple power attacks, the cryptographic research community has developed significantly more advanced attack methods. The idea behind most algorithms remains to perform a statistical analysis by correlating the power trace obtained when executing a cryptographic primitive to a key-dependent guess. With the advancements of cryptographic countermeasures, it is not uncommon that sophisticated (higher-order) power attacks require computation on many millions of power traces in order to find the desired correlation.
In this paper, we study the computational aspects of calculating the most widely used correlation coefficient: the Pearson product-moment correlation coefficient. We study various time-memory trade-off techniques which apply specifically to the cryptologic setting and present methods to extend already completed computations using incremental versions. Moreover, we show how this technique can be applied to second-order attacks, reducing the attack cost significantly when adding new traces to a dataset. We also present methods which allow one to split the potentially huge trace set into smaller, more manageable chunks in order to reduce the memory requirements. Our concurrent implementation of these techniques highlights the benefits of this approach as it allows efficient computations on power measurements consisting of hundreds of gigabytes on a single modern workstation.

Keywords: Side-channel analysis, CPA, Pearson correlation coefficient, higher-order attacks.

1 Introduction

Since the late '90s it is publicly known that the (statistical) analysis of a power trace obtained when executing a cryptographic primitive might correlate to, and hence reveal information about, the secret key material used [13]. This active research area has seen a lot of development over the years: from successfully retrieving the secret key material by inspecting a single power trace using simple power analysis through differential power analysis [13] to using several time samples [17, 29, 11] (cf. [14] for a survey on this topic). Eventually this led to the more general correlation power analysis [6] which might need to inspect a large amount of traces in order to reveal correlations between the power consumption and the bits of the secret key. Moreover, the complexity of the attacks used have increased to higher (n^{th}) order analysis [17]. In this setting one computes joint statistical properties of the power consumption of n samples in order to try and deduce information about the bits of the secret key material used.

This has led to a significant increase in the number of correlation coefficients one has to compute. Researchers have reported handling millions [20, 24, 19], ten million [4], a hundred million [21] and up to three hundred million power traces [3]. This increase of available data requires more processing power in order to compute the required statistical analysis. This is illustrated by recent work: e.g. [1, 18] investigates the usage of graphics processing units to accelerate the computation while [16] explores the sophisticated usage of high-performance computing in this setting. Although many hardware designers and power analysis experts

⋆ This work was done while the first author was an intern in the innovation center crypto & security at NXP Semiconductors.

report whether the implementations of their techniques withstand (higher-order) correlation attacks, it is often not reported how much time was required to mount such attacks.

Over the last decade correlation power analysis (CPA) has become widely used and preferred over differential power analysis since it performs better and requires fewer traces [5]: both important characteristics for measuring the effectiveness of an attack. One of the main ingredients in algorithms based on CPA is the computation of the correlation coefficient. In this paper we take a closer look at the Pearson product-moment correlation coefficient [23], simply referred to as Pearson's ρ, from a computational point of view, taking its role in cryptology into account. Although other correlation coefficients have been considered as well (e.g. in [2]), Pearson's ρ has been used in almost all correlation power attacks over the last decade, since it allows for an efficient correlation computation.

There are a lot of time-memory trade-off techniques one can apply to reduce the computational cost of Pearson's ρ in the cryptologic setting. We study some of these techniques in detail explore the available optimization options when computing correlations in the setting of higher-order attacks. Additionally, we investigate the setting where additional traces are added after an initial CPA attack did not reveal any leakage and we present algorithms which can re-use computations in both the first and second order setting. This study of computational aspects is complemented with techniques allowing the computation of Pearson's ρ on memory constrained devices or, viewed differently, to compute the correlation function on huge data sets using conventional desktop machines. We demonstrate the effectiveness of our approach by showing implementation results of the methods used.

2 Preliminaries

Let X, Y be random variables with values uniformly distributed over $\mathbb{R}$, let x_i, y_i, where $1 \leq i \leq n$, be n samples of these variables, respectively. In this setting the expected value is denoted as $E[X] = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, the covariance as $\sigma(X,Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$, the variance as $\mathrm{Var}(X) = \sigma(X,X) = \sigma_X^2 = E[(X - E[X])^2] = E[X^2] - E[X]^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - \bar{x}^2$, and the standard deviation as $\sigma_X = \sqrt{\mathrm{Var}(X)}$.

The typical attack scenario we are concerned with is as follows. We assume we have selected a specific implementation of a cryptographic algorithm on some device we wish to attack. Let $I(Z,k)$ represent a target intermediate state of the algorithm with input Z and where only a small portion of the secret key is used, denoted by k. We assume that the power consumption of the device at state $I(Z,k)$ is the sum of a data dependent component and some random noise, i.e. $\mathcal{L}(I(Z,k)) + \delta$, where the function $\mathcal{L}(s)$ returns the power consumption of the device during state s, and δ denotes some leakage noise. Following [15], we assume that the noise is random, independent from the intermediate state and is normally distributed with zero mean. The goal of an attacker is to recover the part of the key k by comparing some real power measurements of the device with an estimation of the power consumption under all possible hypothesis for k. However, the adversary does not know the exact leakage function $\mathcal{L}$. Estimating the leakage function is commonly done using the Hamming weight model: in this model it is assumed that there is a relationship between the number of bits set, the Hamming weight or population count, and the power consumption. In a CPA attack, one tries to correlate the real power consumption with the predicted one in order to find the correct portion k of the secret key.

Table 1. Three different strategies to compute $C_{i,j} = \rho(\mathbf{T}_i, \mathbf{K}_j)$ for all $1 \leq i \leq t$, $1 \leq j \leq k$ as is common in first order CPA. The three versions increasingly use the time-memory trade-off paradigm. The memory requirement in parentheses for version 2 is in the case where the input cannot be overwritten.

	Pseudocode	add/sub	mul	div	sqrt	mem
version 0	for $1 \leq i \leq t$ do					
	$\bar{t} = \text{mean}(\mathbf{T}_i)$	t(n-1)		t		1
	for $1 \leq j \leq k$ do					
	$\bar{k} = \text{mean}(\mathbf{K}_j)$	kt(n-1)		kt		1
	$C_{i,j} = \rho_1(\mathbf{T}_i, \bar{t}, \mathbf{K}_j, \bar{k})$	kt(5n-3)	kt(3n+1)	kt	kt	6
	Approximate total	6nkt	3nkt	2kt	kt	8
version 1	for $1 \leq i \leq k$ do $\bar{k}_i = \text{mean}(K_i)$	k(n-1)		k		k
	for $1 \leq i \leq t$ do					
	$\bar{t} = \text{mean}(\mathbf{T}_i)$	t(n-1)		t		1
	for $1 \leq j \leq k$ do					
	$C_{i,j} = \rho_1(\mathbf{T}_i, \bar{t}, \mathbf{K}_j, \bar{k}_j)$	kt(5n-3)	kt(3n+1)	kt	kt	6
	Approximate total	5nkt	3nkt	kt	kt	k
version 2	for $1 \leq i \leq k$ do					
	$\bar{k}_i = \text{mean}(\mathbf{K}_i)$, $\hat{k}_i = 0$	k(n-1)		k		k
	for $1 \leq j \leq n$ do					
	$\mathbf{k}_{i,j} = \mathbf{k}_{i,j} - \bar{k}_i$	kn				(kn)
	$\hat{k}_i = \hat{k}_i + \mathbf{k}_{i,j}^2$	kn	kn			k
	for $1 \leq i \leq t$ do					
	$\bar{t} = \text{mean}(\mathbf{T}_i)$	t(n-1)		t		1
	for $1 \leq j \leq k$ do					
	$C_{i,j} = \rho_2(\mathbf{K}_j, \hat{k}_j, \mathbf{T}_i, \bar{t})$	kt(3n-2)	kt(2n+1)	kt	kt	4
	Approximate total	3nkt	2nkt	kt	kt	2k (kn)

A power trace denotes the power consumption of the device at t points in time during a cryptographic operation. The adversary usually monitors a large amount of such traces in order to cancel the effect of the noise δ. Typically this is done while setting k to a single byte of the key in order to limit the number of enumerations to $2^8 = 256$. As an example, when targeting the Advanced Encryption Standard [8] (AES) it is common to target the intermediate state after the S-box of the first round in the Hamming weight model. Hence, given a plain-text byte p one computes the estimated power consumption of 2^8 key guesses as $\text{PM}(S(p \oplus i))$ where the power model function PM computes the Hamming weight, the function S is the AES S-box and $0 \leq i < 2^8$.

More specifically, let T be a measurement consisting of n traces where each trace recorded data at t different time points: hence, T consist of elements $\mathbf{t}_{i,j} \in \mathbb{R}$ for $1 \leq i \leq t$ and $1 \leq j \leq n$. With $\mathbf{T}_i$ we denote a finite sequence of length n, $\mathbf{T}_i = [\mathbf{t}_{i,1}, \mathbf{t}_{i,2}, \ldots, \mathbf{t}_{i,n}]$, which consists of a single measurement of each trace sampled at time point i. Similarly, we define the matrix K of k estimated power consumption of key guesses consisting of values $\mathbf{k}_{i,j} \in \mathbb{R}$ with $1 \leq i \leq k$ and $1 \leq j \leq n$ corresponding to these n traces.

$$T = \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \\ \vdots \\ \mathbf{T}_t \end{bmatrix} = \begin{bmatrix} \mathbf{t}_{1,1}\ \mathbf{t}_{1,2} \ldots \mathbf{t}_{1,n} \\ \mathbf{t}_{2,1}\ \mathbf{t}_{2,2} \ldots \mathbf{t}_{2,n} \\ \vdots\ \ \vdots\ \ \ddots\ \ \vdots \\ \mathbf{t}_{t,1}\ \mathbf{t}_{t,2} \ldots \mathbf{t}_{t,n} \end{bmatrix}, \quad K = \begin{bmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \\ \vdots \\ \mathbf{K}_k \end{bmatrix} = \begin{bmatrix} \mathbf{k}_{1,1}\ \mathbf{k}_{1,2} \ldots \mathbf{k}_{1,n} \\ \mathbf{k}_{2,1}\ \mathbf{k}_{2,2} \ldots \mathbf{k}_{2,n} \\ \vdots\ \ \vdots\ \ \ddots\ \ \vdots \\ \mathbf{k}_{k,1}\ \mathbf{k}_{k,2} \ldots \mathbf{k}_{k,n} \end{bmatrix} \tag{1}$$

3 Pearson's ρ and Time-Memory Trade-Off Techniques

The Pearson product-moment correlation coefficient, or simply Pearson ρ [23], is a well-known method to measure the linear dependence between two random variables. This correlation is measured as a value between -1 (total negative correlation) and $+1$ (total positive correlation) where a value of zero indicates no correlation. The formula for this correlation is

$$\rho(X,Y) = \frac{\sigma(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\bar{x})(Y-\bar{y})]}{\sigma_X \sigma_Y}, \tag{2}$$

and can be computed, using the notation from Section 2, as

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}. \tag{3}$$

There are various strategies to compute Eq. (3). In statistics, the typical setting is to compute $\rho(X,Y)$ once or possibly compute multiple correlation coefficients where every computation involves different datasets X and Y. In the setting of CPA, this situation is different: one wants to correlate the n trace-values at a fixed point in time with the k different key-guesses (consisting of a vector of n elements each). Hence, one computes the k correlation coefficients where one of the inputs to ρ is fixed. This allows one to perform precomputations in order to lower the number of arithmetic operations (at the cost of additional memory).

In Table 1 we outline three approaches based on the time-memory trade-off paradigm. We assume that, as is the case when performing a first order CPA, we want to compute $t \cdot k$ correlations as $C_{i,j} = \rho(\mathbf{T}_i, \mathbf{K}_j)$ for all $1 \leq i \leq t$, $1 \leq j \leq k$. Table 1 states the computational costs, expressed in terms of additions/subtractions, multiplications, divisions and square root computations required. The approximate total only shows the accumulated part of the largest terms where we assume that $n > t > k$ (which is typical for CPA).

Version 0 uses no additional memory. For efficiency reasons the means of the $\mathbf{T}_i$ are computed once before entering the second for-loop and re-used. The ρ_1 function is identical to Eq. (3) with the means precomputed

$$\rho_1(\mathbf{T}, \bar{t}, \mathbf{K}, \bar{k}) = \frac{\sum_{i=1}^{n}(\mathbf{t}_i-\bar{t})(\mathbf{k}_i-\bar{k})}{\sqrt{\sum_{i=1}^{n}(\mathbf{t}_i-\bar{t})^2}\sqrt{\sum_{i=1}^{n}(\mathbf{k}_i-\bar{k})^2}}.$$

Version 1 precomputes the means of the k different keys, significantly reducing the number of loads from memory and the number of addition and divisions. When $k = 2^8$, a common choice when targeting an individual byte of the secret key, this is a cheap way, in terms of memory, to gain performance. Compared to version 0, the number of additions in the most significant term has been reduced by nkt and we only need half of the divisions.

Version 2 extends the precomputations and computes both $(x_i - \bar{x})$ and $\sum_{i=1}^{n}(x_i-\bar{x})^2$ for all the k different key-guesses, where $\mathbf{k}_{i,j}$ acts as the x_i. When overwriting existing input memory, which effectively centers the samples around zero, this can be computed by only storing k more items. However, when the input cannot be destroyed because it might be used in subsequent computations, the additional memory needed to store all the intermediate values is significant and exceeds kn items. Compared to version 1 the number of additions is further reduced by $2nkt$ (halving the number of additions compared to version 0) while the number of multiplications has been reduced to approximately $2nkt$ (a reduction by nkt

Table 2. Two different strategies to compute $C_{i,j} = \rho(\mathbf{T}_i, \mathbf{K}_j)$ for all $1 \le i \le t$, $1 \le j \le k$ based on the incremental correlation formula (see Eq. (4)) and exploiting the time-memory trade-off paradigm.

	Pseudocode	add/sub	mul	div	sqrt	mem
version 3	for $1 \le i \le t$ do					
	$s_1 = \mathbf{t}_{i,1}, s_2 = \mathbf{t}_{i,1}^2$		t			2
	for $2 \le \ell \le n$ do					
	$s_1 = s_1 + \mathbf{t}_{i,\ell}$	t(n-1)				
	$s_2 = s_2 + \mathbf{t}_{i,\ell}^2$	t(n-1)	t(n-1)			
	for $1 \le j \le k$ do					
	$s_3 = \mathbf{k}_{j,1}, s_4 = \mathbf{k}_{j,1}^2$		kt			2
	$s_5 = \mathbf{t}_{i,1}\mathbf{k}_{j,1}$		kt			1
	for $2 \le \ell \le n$ do					
	$s_3 = s_3 + \mathbf{k}_{j,\ell}$	kt(n-1)				
	$s_4 = s_4 + \mathbf{k}_{j,\ell}^2$	kt(n-1)	kt(n-1)			
	$s_5 = s_5 + \mathbf{t}_{i,\ell}\mathbf{k}_{j,\ell}$	kt(n-1)	kt(n-1)			
	$C_{i,j} = \rho_3(s_1, s_2, s_3, s_4, s_5)$	3kt	7kt	kt	kt	3
	Approximate total	3nkt	2nkt	kt	kt	8
version 4	for $1 \le j \le k$ do					
	$\mathbf{s}_{j,3} = \mathbf{k}_{j,1}, \mathbf{s}_{j,4} = \mathbf{k}_{j,1}^2$		k			2k
	for $2 \le \ell \le n$ do					
	$\mathbf{s}_{j,3} = \mathbf{s}_{j,3} + \mathbf{k}_{j,\ell}$	k(n-1)				
	$\mathbf{s}_{j,4} = \mathbf{s}_{j,4} + \mathbf{k}_{j,\ell}^2$	k(n-1)	k(n-1)			
	for $1 \le i \le t$ do					
	$s_1 = \mathbf{t}_{i,1}, s_2 = \mathbf{t}_{i,1}^2$		t			2
	for $2 \le \ell \le n$ do					
	$s_1 = s_1 + \mathbf{t}_{i,\ell}$	t(n-1)				
	$s_2 = s_2 + \mathbf{t}_{i,\ell}^2$	t(n-1)	t(n-1)			
	for $1 \le j \le k$ do					
	$s_5 = \mathbf{t}_{i,1}\mathbf{k}_{j,1}$		kt			1
	for $2 \le \ell \le n$ do					
	$s_5 = s_5 + \mathbf{t}_{i,\ell}\mathbf{k}_{j,\ell}$	kt(n-1)	kt(n-1)			
	$C_{i,j} = \rho_3(s_1, s_2, \mathbf{s}_{j,3}, \mathbf{s}_{j,4}, s_5)$	3kt	7kt	kt	kt	3
	Approximate total	nkt	nkt	kt	kt	2k

multiplications). When overwriting existing memory x_i with $x_i - \bar{x}$ the Pearson ρ can be computed as

$$\rho_2(X, \hat{x}, Y, \bar{y}) = \frac{\sum_{i=1}^{n} x_i \cdot (y_i - \bar{y})}{\sqrt{\hat{x} \cdot \sum_{i=1}^{n} (y_i - \bar{y})^2}}.$$

4 Incremental Pearson

It is well-known that Eq. (3) can be rewritten as

$$\rho(X, Y) = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \sqrt{n \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}} \tag{4}$$

by computing the means directly (substituting $\bar{x}$ with $\frac{1}{n}\sum_{i=1}^{n} x_i$) and performing some reordering. This alternative formula allows for different precomputation strategies and, moreover, has additional benefits which are discussed in this section.

In a similar vein to the approach taken in Section 3 we analyse two different versions based on the time-memory trade-off paradigm. The two approaches (version 3 and version

4) discussed here and which compute Eq. (4) are summarized in Table 2. This approach essentially needs to compute the five values

$$s_1 = \sum_{i=1}^{n} x_i, \quad s_2 = \sum_{i=1}^{n} x_i^2, \quad s_3 = \sum_{i=1}^{n} y_i, \quad s_4 = \sum_{i=1}^{n} y_i^2, \quad s_5 = \sum_{i=1}^{n} x_i y_i.$$

Once these have been computed Eq. (4) can be computed as

$$\rho_3(s_1, s_2, s_3, s_4, s_5) = \frac{(ns_5 - s_1 s_3)}{\sqrt{(ns_2 - s_1^2)(ns_4 - s_3^2)}}.$$

Version 3 in Table 2 computes ρ_3 without using additional storage where $\mathbf{t}_{i,j}$ acts as the x_i and $\mathbf{k}_{i,j}$ as y_i. For a fixed $\mathbf{T}_i$ the values of s_1 and s_2 are computed at the beginning of the for-loop and re-used for the multiple key guesses. However, the values s_3, s_4 and s_5 are recomputed. Version 4 also pre-computes s_3 and s_4 at the cost of storing $2k$ data-elements. This reduces the number of required additions by a factor three to nkt and the number of multiplications by a factor two to nkt. It does not make sense to also pre-compute $s_5 = \sum_{i=1}^{n} x_i y_i$ since these results cannot be re-used; this would only costs additional storage without reducing the number of arithmetic operations.

Incremental first-order CPA. In addition to being computationally more efficient than the fastest "standard version" variant (cf. the performance data of version 2 in the left plot of Figure 4) Eq. (4) has another interesting feature. Suppose we perform a CPA attack on a trace set of size n, but are not able to observe any significant leakage. In the hope of observing some leakage, an adversary could try and add m additional traces to the measurement. By using any variant of the "standard version" (version 0, version 1, or version 2), the attacker must first perform his attack on the set of n traces, then on the larger set containing $n + m$ traces, duplicating many arithmetic computations. Using a variation of version 4, the adversary can compute the correlations on the first trace set, but then extend the computations to compute a CPA on the extended set of $n + m$ traces without using a significant amount of additional memory. Indeed, Eq. (4) only requires to compute sums of elements of vectors, which can be stored and later updated by adding the elements of the new traces. This approach is infeasible using, for instance, version 2.

Let us consider two variants of version 4. In version 5, the $2t$ values $\mathbf{s}_{i,1} = \sum_{i=1}^{n} x_i$ and $\mathbf{s}_{i,2} = \sum_{i=1}^{n} x_i^2$ are precomputed and reused when adding more traces. Similarly, we extend version 5 to a version 6 where we also precompute and reuse the tk values $\mathbf{s}_{i,5} = \sum_{i=1}^{n} x_i y_i$. Then the cost of computing the correlation coefficients using n traces first and subsequently with $m+n$ traces is summarized in the following table where only the largest terms are shown.

	version 2	version 5	version 6
Approximate #add	3kt(2n+m)	kt(2n+m)	kt(m+n)
Approximate #mul	2kt(2n+m)	kt(2n+m)	kt(n+m)
Approximate #mem	kn	2(k+t)	kt

Here we assume that version 2 is the high-memory version that cannot replace the input values in-place as discussed in Section 3. For example, assume the scenario where $m = n$. In this case, version 2 requires approximately $9ktn$ additions and $6ktn$ multiplications, as well as a rather large amount of storage of $2kn$ elements. At the cost of storing $2(k + t)$ elements, version 5 reduces the number of additions by a factor three to $3ktn$ and the number

of multiplications by a factor two to $3ktn$ (compared to version 2). If one can spare to store kt data elements the cost of version 2 is even further reduced: the number of additions by a factor 4.5 to $2ktn$ and the number of multiplications with a factor three to $2ktn$. Although these are only constant improvements, they can make a significant difference in practice when computing CPA on large datasets where both t and n can be very large.

5 Computational Aspects of Second Order CPA

A common measure to protect against side-channel attacks is known as masking (see e.g. [7, 10, 26]) where the secret data is split into multiple shares. One approach to attack such schemes is to combine these shares in order to find a correlation between the power consumption of the combination of these shares and the secret material: this is known as higher-order power analysis (see e.g. [17, 29, 11]). In order to combine two power measurements $\mathbf{T}_1 = [\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,n}]$ and $\mathbf{T}_2 = [\mathbf{t}_{2,1}, \mathbf{t}_{2,2}, \ldots, \mathbf{t}_{2,n}]$ as $C(\mathbf{T}_1, \mathbf{T}_2) = [\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n]$ one can compute the normalized product (compute $\mathbf{t}_i = (\mathbf{t}_{1,i} - \bar{\mathbf{T}}_1)(\mathbf{t}_{2,i} - \bar{\mathbf{T}}_2)$ for $1 \leq i \leq n$) [7], the absolute difference (compute $\mathbf{t}_i = |\mathbf{t}_{1,i} - \mathbf{t}_{2,i}|$ for $1 \leq i \leq n$) [17] or the sum (compute $\mathbf{t}_i = \mathbf{t}_{1,i} + \mathbf{t}_{2,i}$ for $1 \leq i \leq n$) [28]. As shown in [25, 28] the normalized product approach performs best when Pearson's correlation coefficient is used and assuming the Hamming weight leakage model. Following these conclusions we focus exclusively on this combining function in this section in the setting of second order CPA.

Assuming that we also consider the combination of a power measurement with itself, there are in total $\sum_{i=1}^{t} i = t(t+1)/2$ possible unique pairs from the t power measurements. In practice, however, one does not necessarily combine all possible pairs. The computation of the shares usually occurs within a bounded time span. Therefore we introduce another parameter when computing higher order attacks, the window size w. This positive integer is an estimation of the maximum distance between two shares. In the algorithms considered in this section, given t power measurements and a window size $w \leq t$ $W = W(t, w) = (t - w)w + \sum_{i=1}^{w} i = w(t - \frac{w-1}{2})$ denotes the number of pairs considered. The smaller the window w, the fewer computations one has to perform; note that setting $w = t$ is equivalent to computing on all possible pairs. In the following, similar to the analysis in Sections 3 and 4, we describe three basic algorithms to perform second-order CPA and list various time-memory trade-off techniques that can be applied. We exclusively focus on optimizations improving the most significant term in the computational cost and do not consider various minor optimizations one can achieve as well.

5.1 Naive Second Order CPA: Version 7

Table 3 outlines the "naive" version of a second-order CPA. This version does not use any auxiliary memory except one vector of n elements which holds the result of the combining function $C(\mathbf{T}_1, \mathbf{T}_2)$. Following the approach from Section 3 one can reduce the number of arithmetic operations at the cost of using additional memory by

1. precomputing the k means of $\mathbf{K}_i$, at a cost of approximately kn additions and k divisions,
2. precomputing the k variances of $\mathbf{K}_i$, at a cost of an additional kn additions, kn subtractions and kn multiplications,
3. if the input can be overwritten, replace the $\mathbf{k}_{i,j}$ by $\mathbf{k}_{i,j} - \bar{k}_i$, otherwise store the nk normalized $\mathbf{k}_{i,j}$ values,

Table 3. Second-order CPA with window w and normalized product combining based on Eq. (2).

	Pseudocode	add/sub	mul	div	sqrt	mem
version 7	for $1 \leq i \leq t$ do					
	$\bar{t}_i = \text{mean}(\mathbf{T}_i)$	t(n-1)		t		1
	for $i \leq j \leq \min(t, i+w)$ do					
	$\bar{t}_j = \text{mean}(\mathbf{T}_j)$	W(n-1)		W		1
	$\bar{s} = 0$					1
	for $1 \leq \ell \leq n$ do					
	$\mathbf{s}_\ell = (\mathbf{t}_{i,\ell} - \bar{t}_i)(\mathbf{t}_{j,\ell} - \bar{t}_j)$	2nW	nW			n
	$\bar{s} = \bar{s} + \mathbf{s}_\ell$	nW				
	$\bar{s} = \bar{s}/n$			W		
	for $1 \leq \ell \leq k$ do					
	$\bar{k} = \text{mean}(\mathbf{K}_\ell)$	kW(n-1)		kW		
	$C_{i,j,\ell} = \rho_1(\mathbf{K}_\ell, \bar{k}, \mathbf{s}, \bar{s})$	kW(5n-3)	kW(3n+1)	kW	kW	6
	Approximate total	6nkW	3nkW	2kW	kW	n

By using optimizations 1 and 2, we obtain a first variant of version 7 (denoted version 7a) which requires a medium amount of memory: this version precomputes $2k$ additional values which significantly reduces the computational cost, reducing the number of addition by a factor 1.5 (from $6nkW$ to $4nkW$) and the number of multiplications by a factor 1.5 (from $3nkW$ to $2nkW$).

Furthermore, by also applying optimization 3, we obtain the fastest version (denoted version 7b). This might come at no additional cost in terms of memory if the input can be overwritten or, if this is not an option, requires storage for nk additional values. This final trade-off allows us to save another nkW subtractions: a factor two improvement over version 7.

5.2 Naive Second Order CPA using Eq. (4): Version 8

This version of a second order CPA modifies version 7 such that it uses Eq. (4) to compute the individual correlation coefficients; it is outlined in Table 4. Using this approach, we can directly benefit from the computation of the temporary vector $\mathbf{s} = C(\mathbf{T}_1, \mathbf{T}_2)$ to precompute some values required in Eq. (4) with almost no additional computational cost. Analogue to version 7, the version presented in Table 4 requires almost no additional memory with the exception of what is needed to store the temporary vector $\mathbf{s}$. One additional optimization improving the most significant term can be applied based on the time-memory trade-off paradigm

1. precompute the $2k$ values $s_3 = \sum_{i=1}^{n} \mathbf{k}_{\ell,i}$ and $s_4 = \sum_{i=1}^{n} \mathbf{k}_{\ell,i}^2$, at a cost of $2kn$ additions and kn multiplications.

By applying this single optimization, we can improve the computational cost of version 8 at the price of $2k$ additional memory (denoted by version 8a). This version halves the number of multiplication (from $2nkW$ to nkW) and reduces the number of additions by a factor three (from $3nkW$ to nkW).

5.3 Incremental Second Order CPA: Version 9

The techniques used in Section 4 can also be applied to higher-order CPAs. In this section we outline how to achieve this in the second order CPA setting. Using such a version enables one

Table 4. Second-order CPA with window w and normalized product combining based on Eq. (4)

	Pseudocode	add/sub	mul	div	sqrt	mem
version 8	for $1 \le i \le t$ do					
	$\bar{t}_i = \text{mean}(\mathbf{T}_i)$	t(n-1)		t		1
	for $i \le j \le \min(t, i+w)$ do					
	$\bar{t}_j = \text{mean}(\mathbf{T}_j)$	W(n-1)		W		
	$s_1 = 0, s_2 = 0$					2
	for $1 \le \ell \le n$ do					
	$\mathbf{s}_\ell = (\mathbf{t}_{i,\ell} - \bar{t}_i)(\mathbf{t}_{j,\ell} - \bar{t}_j)$	2nW	nW			n
	$s_1 = s_1 + \mathbf{s}_\ell$	nW				
	$s_2 = s_2 + \mathbf{s}_\ell^2$	nW	nW			
	for $1 \le \ell \le k$ do					
	$s_m = 0$ for $3 \le m \le 5$					
	for $1 \le m \le n$ do					
	$s_3 = s_3 + \mathbf{k}_{\ell,m}$	nkW				
	$s_4 = s_4 + \mathbf{k}_{\ell,m}^2$	nkW	nkW			
	$s_5 = s_5 + \mathbf{s}_m \mathbf{k}_{\ell,m}$	nkW	nkW			
	$C_{i,j,\ell} = \rho_3(s_1, s_2, s_3, s_4, s_5)$	3kW	7kW	kW	kW	3
	Approximate total	3nkW	2nkW	kW	kW	n

to compute a second order CPA *incrementally*, resulting in potentially significant savings when adding more power traces to a power measurement. Obviously, the number of intermediate results one needs to keep track of increases in this setting. Instead of the five values in Section 4, one has to store thirteen in this case.

In order to deduce a formula based on these thirteen values, which can be updated incrementally, we replace the variable X with $C(T, U)$: the combination of the two traces T and U. More precisely, given our choice of combining function, we replace the individual x_i with $(t_i - \bar{t})(u_i - \bar{u})$.

$$\rho(C(T,U), Y) = \frac{n\lambda_1 - \lambda_2 s_3}{\sqrt{n\lambda_3 - \lambda_2^2}\sqrt{ns_9 - s_3^2}} = \rho_4(s_1, \ldots, s_{13}) \tag{5}$$

The transformation is obtained by expanding the formula and gathering all the individual terms. This leads to the following values for the variables used in Eq. (5)

$$\begin{aligned}
&\lambda_1 = s_{10} - \frac{s_1 s_7 + s_2 s_5}{n} + \frac{s_1 s_2 s_3}{n^2}, && \lambda_2 = s_4 - \frac{s_1 s_2}{n},\\
&\lambda_3 = s_{11} - \frac{2s_2 s_{12} + 2s_1 s_{13}}{n} + \frac{s_2^2 s_6 + 4 s_1 s_2 s_4 + s_1^2 s_8}{n^2} - \frac{3(s_1 s_2)^2}{n^3},\\
&s_1 = \textstyle\sum_{i=1}^n t_i, && s_2 = \textstyle\sum_{i=1}^n u_i, && s_3 = \textstyle\sum_{i=1}^n y_i,\\
&s_4 = \textstyle\sum_{i=1}^n t_i u_i, && s_5 = \textstyle\sum_{i=1}^n t_i y_i, && s_6 = \textstyle\sum_{i=1}^n t_i^2,\\
&s_7 = \textstyle\sum_{i=1}^n u_i y_i, && s_8 = \textstyle\sum_{i=1}^n u_i^2, && s_9 = \textstyle\sum_{i=1}^n y_i^2,\\
&s_{10} = \textstyle\sum_{i=1}^n t_i u_i y_i, && s_{11} = \textstyle\sum_{i=1}^n t_i^2 u_i^2, && s_{12} = \textstyle\sum_{i=1}^n t_i^2 u_i,\\
&s_{13} = \textstyle\sum_{i=1}^n t_i u_i^2.
\end{aligned}$$

Table 5 outlines the approach to implement Eq. (5) with some straightforward improvements, where the $\mathbf{t}_{i,m}$ acts as the t_i, the $\mathbf{t}_{j,m}$ acts as the u_i and the $\mathbf{k}_{\ell,m}$ as y_i. Again, we can reduce the number of operations by using more memory. We distinguish the following optimizations

1. precompute the $2k$ values $s_3 = \sum_{i=1}^n y_i$ and $s_9 = \sum_{i=1}^n y_i^2$, which requires approximately $2kn$ additions and kn multiplications.

2. precompute the kt values $s_5 = \sum_{i=1}^{n} t_i y_i$. This precomputation requires approximately ktn additions and ktn multiplications.

A faster variant of version 9 can be obtained by applying optimization 1 (denoted version 9a) for an additional storage cost of $2k$ values. This optimization reduces the number of additions by a factor $\frac{5}{3}$ from $5nkW$ to $3nkW$ and the number of multiplications by a factor $\frac{4}{3}$ from $4nkW$ to $3nkW$. By also including optimization 2 (denoted version 9b) we apply the time-memory trade-off even further, requiring to store tk additional elements. The effect is significant, as not only the value s_5 is no longer computed in the innermost loop, but the value s_7 now comes for free. Thus, in comparison to version 9, the number of additions is reduced to $kn(t + W)$ (which for large values of W means a reduction by a factor five) while the number of multiplications is reduced to $kn(t + 2W)$ (which for large values of W means a reduction by a factor two).

An overview of the cost of the various second order algorithms is given below.

	Approx. #add	Approx. #mul	Approx. #mem
Version 7	6nkW	3nkW	n
Version 7a	4nkW	2nkW	n
Version 7b	3nkW	2nkW	n or nk
Version 8	3nkW	2nkW	n
Version 8a	nkW	nkW	n
Version 9	5nkW	4nkW	14
Version 9a	3nkW	3nkW	2k
Version 9b	nk(t+W)	nk(t+2W)	kt

Incremental Second Order CPA. As stated in Section 5.3, version 9 has the potential to reuse computations when adding more power traces to the power measurement. Indeed, keeping track of a few values allows for a much more efficient computation of the correlation coefficients when adding a batch of new traces to the power measurement. We consider the setting similar to the one in Section 4: the attacker first performs a second order CPA on a set of n traces. Next, an additional m traces are added to the power measurement and a computation on all the $m + n$ traces is performed. Let us consider a variant of version 9 (denoted version 9c) consisting of an extension of version 9b in which we also store the kW values

$$s_{10,i,j,\ell} = \sum_{a=1}^{m+n} \mathbf{t}_{i,a}\mathbf{t}_{j,a}\mathbf{k}_{\ell,a}, \quad \text{for } 1 \leq i \leq t, \quad i \leq j \leq \min(t, i + w), \quad \text{and } 1 \leq \ell \leq k.$$

The total cost of computing the kW correlation coefficients using first n and then $n + m$ traces with several attack variants is summarized in the following table.

	version 7b	version 8a	version 9b	version 9c
Approx. #add	3kW(2n+m)	kW(2n+m)	k(t+W)(2n+m)	k(t+W)(n+m)
Approx. #mul	2kW(2n+m)	kW(2n+m)	k(t+2W)(2n+m)	k(t+2W)(n+m)
Approx. #mem	k(n+m)	n+m	kt	kW

Considering a scenario where $m = n$, i.e. in which an attacker adds n new traces to a set of n traces (working with $2n$ traces in total), version 7b requires $9nkW$ additions and $6nkW$ multiplications, and requires memory for $2kn$ elements, as the values cannot be modified in-place just as in the first order setting. Using much less memory, version 8a can compute this

Table 5. Second-order CPA with window w and normalized product combining based on Eq. (5). Note that the operation count for ρ_4 is an upper bound. This count can be reduced by simplifying Eq. (5). However, this does not reflect on the most significant term.

	Pseudocode	add/sub	mul	div	sqrt	mem
version 9	for $1 \le i \le t$ do					
	$s_1 = 0, s_6 = 0$					2
	for $1 \le j \le n$ do					
	$s_1 = s_1 + \mathbf{t}_{j,i}$	nt				
	$s_6 = s_6 + \mathbf{t}^2_{j,i}$	nt	nt			
	$s_l = 0$ for $l \in \{2, 4, 8, 11, 12, 13\}$					6
	for $i \le j \le \min(t, i + w)$ do					
	for $1 \le \ell \le n$ do					
	$s_2 = s_2 + \mathbf{t}_{\ell,j}$	nW				
	$s_8 = s_8 + \mathbf{t}^2_{\ell,j}$	nW	nW			
	$s_4 = s_4 + \mathbf{t}_{\ell,i}\mathbf{t}_{j,\ell}$	nW	nW			
	$s_{12} = s_{12} + \mathbf{t}^2_{\ell,i}\mathbf{t}_{\ell,j}$	nW	nW			
	$s_{13} = s_{13} + \mathbf{t}_{\ell,i}\mathbf{t}^2_{\ell,j}$	nW	nW			
	$s_{11} = s_{11} + \mathbf{t}^2_{\ell,i}\mathbf{t}^2_{\ell,j}$	nW	nW			
	for $1 \le \ell \le k$ do					
	$s_l = 0$ for $l \in \{3, 5, 7, 9, 10\}$					5
	for $1 \le m \le n$ do					
	$s_3 = s_3 + \mathbf{k}_{m,\ell}$	nkW				
	$s_9 = s_9 + \mathbf{k}^2_{m,\ell}$	nkW	nkW			
	$s_5 = s_5 + \mathbf{t}_{m,i}\mathbf{k}_{m,\ell}$	nkW	nkW			
	$s_7 = s_7 + \mathbf{t}_{m,j}\mathbf{k}_{m,\ell}$	nkW	nkW			
	$s_{10} = s_{10} + \mathbf{t}_{m,i}\mathbf{t}_{m,j}\mathbf{k}_{m,\ell}$	nkW	nkW			
	$C_{i,j,\ell} = \rho_5(s_1, \ldots, s_{13})$	13kW	24kW	7kW	kW	7
	Approximate total	5nkW	4nkW	7kW	kW	20

second order CPA using $3nkW$ additions and $3nkW$ multiplications, resulting in a reduction of a factor three in the number of additions and a factor two in the number of multiplications compared to version 7b. Version 9b slightly increases the computational cost of version 8a, but requires kt memory, which in practice is generally (much) smaller than $2n$ (typically $k = 2^8$ and $t \approx 10^3$ while $n \approx m > 10^7$). Finally, version 9c reduces the number of additions and multiplications even further by a factor 1.5 with respect to version 9b: this comes at the cost of using less than w times more memory than version 9b (since $kW < kwt$).

6 Memory Constrained CPA

Reducing the computation time when computing the correlation coefficient is just one important characteristic in a state-of-the-art CPA implementation. Due to the common practice of collecting a large number of samples per trace (the variable t) and collecting a large number of traces (the variable n), values around $n \approx 10^8$ are not uncommon [21, 3], the size of the main memory available in common desktop machines is in most cases insufficient to load the entire measurement at once. In this section, we present some methods to efficiently partition the data under such memory constraints. These methods allow to perform CPA attacks on a large set of power traces on widely available desktop machines and do not significantly increase the computation time (in terms of number of arithmetic operations) compared to the setting where sufficient memory would be available.

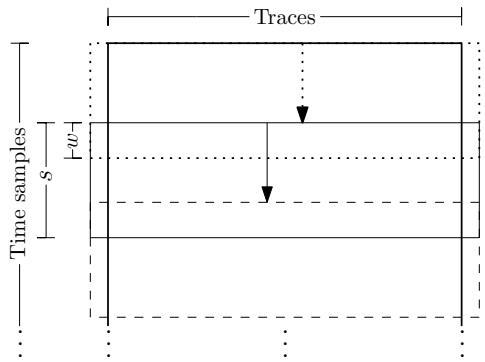

Fig. 1. A sliding window approach when splitting the data across the time samples.

6.1 Split across the Time Samples

The most natural, and straightforward, way of reducing the memory requirement is to split the measurement across the time samples. The idea is to consider the entire set of traces at once but for a reduced number of time samples at a time. Consider the set of power traces as the matrix T (see the definition from Section 2) consisting of n power traces, each of which consists of t power measurements. Let $s < t$ be the number of time samples such that $s \times n$ elements in the trace file can be loaded into the main memory. When splitting across the time samples, one could load s samples at a time, compute the correlation, then move to the next s samples, and repeat the process until all correlations have been computed.

However, when computing correlations in an h-th order correlation attack scenario, one typically defines a window size $1 \leq w < s$ such that all possible combinations of h time samples within this window are considered. A sliding window approach can be used in order to compute all required combinations while keeping a maximum of s time samples at a time in memory. This approach starts by reading the first s time samples into memory, then at each step loads and analyses the next $(s - w)$ samples in the following way. Starting at the first time sample, we combine it with all the samples within the window size w. We then increase the start position incrementally, ensuring that the last sample considered, starting at position $s - w$, can still read the entire window, until the s-th sample, into memory. After having computed the correlation coefficient of these $w(s - w)$ combinations with the key guesses, the first $(s - w)$ time samples are discarded, the last w samples are moved to the beginning and the next batch of $(s - w)$ samples for all n traces are read until we have covered all t measurements. This approach is illustrated in Figure 1.

6.2 Split across the Traces

Another approach to partition the measurement in order to reduce the memory requirement, is to use a limited number of traces at a time but consider the entire range of time samples. One example of such a partitioning was already discussed in Section 4 and Section 5.3 in the setting of the incremental versions for computing Pearson's correlation coefficient. This approach is illustrated in Figure 2. In this subsection we show how this partitioning can be conducted using the work by Dunlap [9] where it is studied how to combine correlation coefficients from different sets given the means, standard deviation and correlation of these sets.

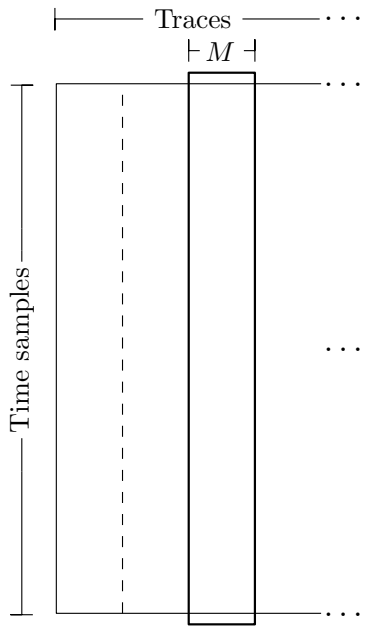

Fig. 2. Loading the entire range of time samples into memory while processing a batch of traces at-a-time.

This approach does not update the value of the correlation coefficient continuously, by adding new traces, but instead computes the correlation coefficient of a set consisting of fewer traces and combines these results to compute the (estimation of the) correlation coefficient when taking all traces into account. This approach also reduces the memory requirement significantly since a much smaller set of traces needs to be kept in memory at a time. Note that in this case, as we have access to the entire range of time samples, we do not need to use the sliding window approach presented in the previous section. Let

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \vdots \\ \mathbf{t}_n \end{bmatrix} \quad \text{and} \quad \mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_n \end{bmatrix}$$

be two vectors that we want to correlate in a correlation power analysis attack. $\mathbf{T}$ represents a vector from the matrix T at a fixed time point and $\mathbf{K}$ denotes a vector from the matrix of estimated power consumption K for a given key guess (as defined in Eq. (1)).

We then split the vector $\mathbf{T}$ into m groups $\mathbf{M}_j$ having respective sizes s_j. Hence we have $\sum_{j=1}^{m} s_j = n$. Similarly, we split the vector $\mathbf{K}$ into m partitions $\mathbf{L}_j$ of sizes s_j. Thus, the sizes of the groups $\mathbf{M}_j$ and $\mathbf{L}_j$ are equal, for all $1 \leq j \leq m$. Now, for each group of traces $\mathbf{M}_j$, we compute the m standard deviations $\sigma_{\mathbf{M}_j}$, the m means $\bar{\mathbf{M}}_j$ and the m correlation coefficients with the corresponding groups from K, $r(\mathbf{M}_j, \mathbf{L}_j)$. The same is done for the m standard deviations $\sigma_{\mathbf{L}_j}$ and the m means $\bar{\mathbf{L}}_j$.

In the following we present formulas which compute directly the final correlation given these values. Note however that in an actual implementation, the individual values should be combined incrementally to save on storage. The sizes s_j of the groups of traces $\mathbf{M}_j$ and $\mathbf{L}_j$ are selected such that the $s_j \cdot t + s_j \cdot k$ measurement values fit into memory.

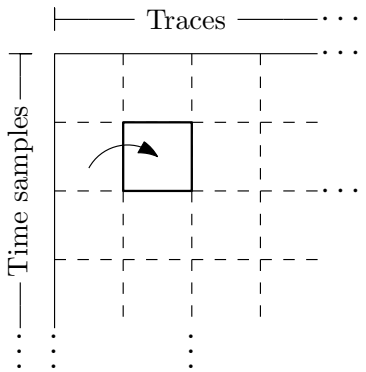

Fig. 3. Combining the approach from Fig. 2 and Fig. 1 into *block-partitioning*. Load part of the range of time samples using part of the traces at-a-time.

In [9], Dunlap present the following result, which we summarize in our setting, to compute the correlation coefficient of the larger trace set based on the mean, standard deviation and correlation coefficients of the smaller sets

$$\rho(\mathbf{T}, \mathbf{K}) = \frac{\sum_{i=1}^{m} s_i \left(\sigma_{\mathbf{M}_i} \sigma_{\mathbf{L}_i} \rho(\mathbf{M}_i, \mathbf{L}_i) + \tilde{\mathbf{M}}_i \tilde{\mathbf{L}}_i \right)}{\sqrt{\sum_{i=1}^{m} s_i \left(\sigma_{\mathbf{M}_i}^2 + \tilde{\mathbf{M}}_i^2 \right)} \sqrt{\sum_{i=1}^{m} s_i \left(\sigma_{\mathbf{L}_i}^2 + \tilde{\mathbf{L}}_i^2 \right)}}. \tag{6}$$

We refer to this approach as exact combining. The $\tilde{\mathbf{M}}_i$ ($\tilde{\mathbf{L}}_i$) are defined as the difference between the mean of the measurements of an individual time sample in $\bar{\mathbf{M}}_i$ ($\bar{\mathbf{L}}_i$) and the mean of all the measurements of this time sample in T (K); i.e.

$$\tilde{\mathbf{M}}_i = \bar{\mathbf{M}}_i - \frac{\sum_{j=1}^{m} s_j \bar{\mathbf{M}}_j}{\sum_{j=1}^{m} s_j}, \quad \tilde{\mathbf{L}}_i = \bar{\mathbf{L}}_i - \frac{\sum_{j=1}^{m} s_j \bar{\mathbf{L}}_j}{\sum_{j=1}^{m} s_j}.$$

Note that Eq. (6) computes *exactly the same correlation coefficient* as if directly computing on the larger trace files using Eq. (3) and is not an approximation.

If it is known that all s_i are equal, i.e. we have evenly split-up n traces in m groups of size n/m, *and* we assume that all means of the individual smaller groups of traces $\mathbf{M}_j$ are equal *and* the means of the smaller groups of key guesses $\mathbf{L}_j$ are equal then Eq. (6) can be simplified (cf. [9]) to

$$\rho(\mathbf{T}, \mathbf{K}) = \frac{\sum_{i=1}^{m} \sigma_{\mathbf{M}_i} \sigma_{\mathbf{L}_i} \rho(\mathbf{M}_i, \mathbf{L}_i)}{\sqrt{\sum_{i=1}^{m} \sigma_{\mathbf{M}_i}^2} \sqrt{\sum_{i=1}^{m} \sigma_{\mathbf{L}_i}^2}}. \tag{7}$$

We refer to this approach as approximate combining. Moreover, if one makes the assumptions that the standard deviations of all the $\mathbf{M}_j$ are also equal *and* that all the standard deviations for all the $\mathbf{L}_j$ are equal, then Eq. (7) simplifies further to

$$\rho(\mathbf{T}, \mathbf{K}) = m^{-1} \sum_{i=1}^{m} \rho(\mathbf{M}_i, \mathbf{L}_i). \tag{8}$$

We refer to this approach as the arithmetic mean.

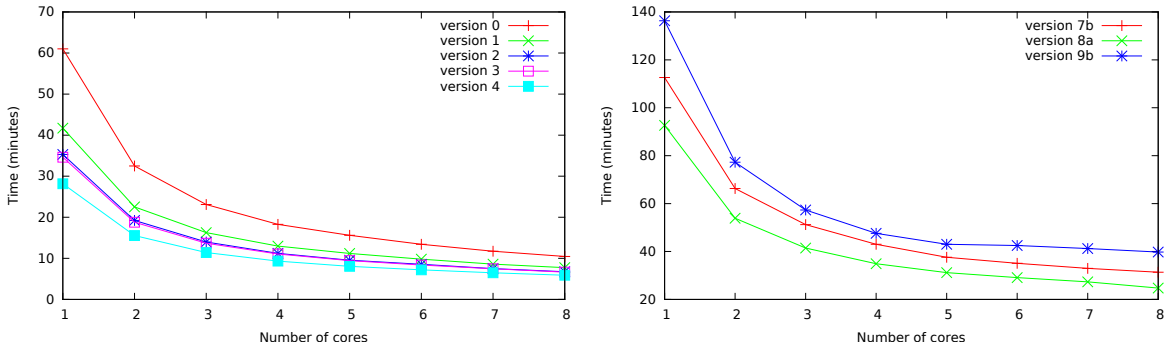

Fig. 4. Performance data when using $n = 10^7$, $k = 2^8$, and $t = 2^9$. The left plot shows first-order CPA results computing 2^{17} correlation coefficients while the plot on the right computes a second-order CPA using $w = 4$ computing $W(t, w) \cdot 256 \approx 2^{19}$ correlation coefficients. The reported timings including the loading time of the data from disk to memory.

6.3 Block Partitioning

One can combine the approaches from Section 6.1 and Section 6.2. Using such a block-partitioning approach, we only consider a subset of the time samples and a subset of the traces. Then, similar to Section 6.1, we compute the correlation coefficients of these subsets, and store them in order to combine them with the correlation factors of the next chunks later. The combining of these correlation coefficients can be done following the approach from Dunlap presented in Section 6.2. This technique allows us to perform correlation attacks on very large data sets even when the available memory is limited. This approach is illustrated in Figure 3.

7 Benchmark Results and Experiments

In order to compare the different approaches and see if the arithmetic instruction counts directly corresponds to performance speedups in practice, we implemented the different versions to compute the Pearson correlation coefficient from Section 3, Section 4 and Section 5. Our implementations allow computations to be computed concurrently using POSIX threads with the pthreads software library [22]. Our benchmark platform is an Intel Xeon (CPU E5-2650 v2), which has eight CPUs, equipped with 32GB of memory. The (parts) of the measurement file which needs to be loaded into memory is stored on a local 256 GB solid state disk (SSD) (Liteonit LCS-256M6S). As a reference we use a power measurement consisting of many time samples, from which we only use a smaller range of $t = 2^9$ samples, and where $n = 10^7$ traces have been collected. Each measurement is represented in an 8-byte double-precision floating-point format. All intermediate computation performed are using double-precision floating-point arithmetic. All algorithms discussed in the previous sections can be made to compute the various correlation coefficients concurrently without too much overhead. By dividing the calculations in multiple threads we can reduce the computation time, taking advantage of the wide availability of multi-core machines.

7.1 First order CPA

In the left plot in Fig. 4 we show the performance results when using parameters $n = 10^7$, $k = 2^8$, and $t = 2^9$: i.e. computing 2^{17} correlation coefficients using 10^7 traces. The correctness of

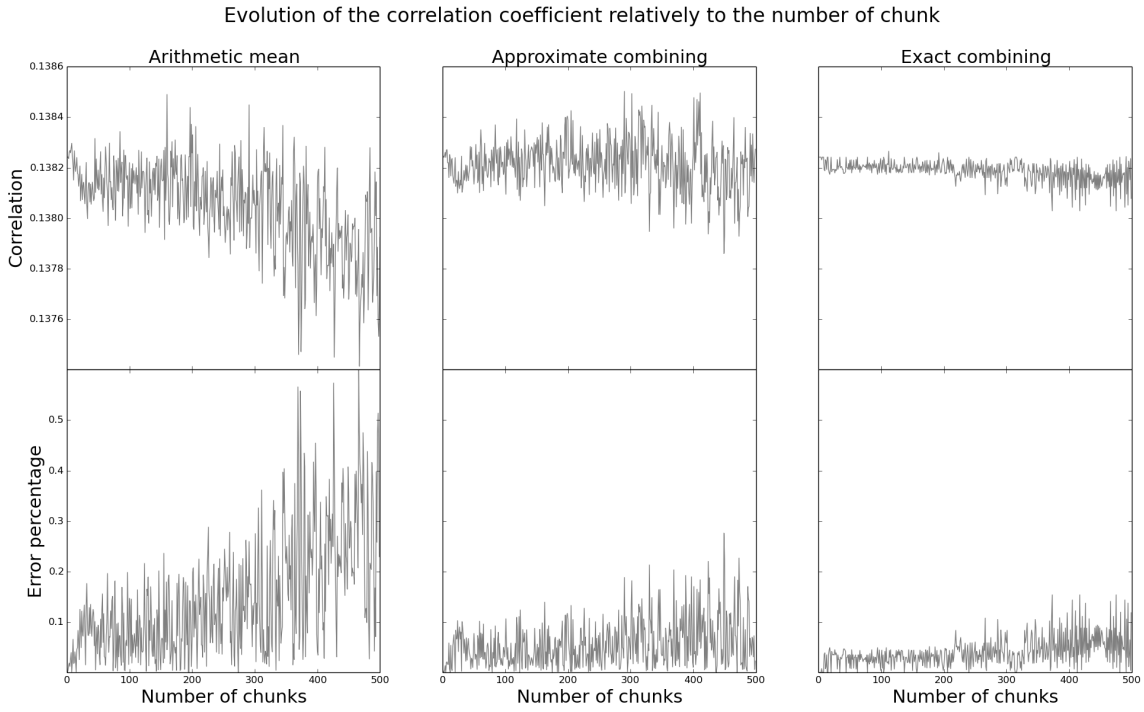

Fig. 5. Experimental results when computing the correlation coefficient using the arithmetic mean (Eq. (8)), approximate combining (Eq. (7)), and exact combining (Eq. (6)) approach. The number of chunks m is varied when using a total number of traces of 10^5. The top graphs show the computed correlation and the bottom graphs the error percentage to the real correlation coefficient.

the implementation is verified with a slow (naive) single-threaded implementation of Pearson's correlation coefficient. The left plot of Fig. 4 shows performance results when varying the number of cores used in the computation for the five different version. These performance numbers are in-line with the arithmetic counts from Table 1 and Table 2. The fastest "naive" version (version 2) performs almost identical compared to version 3. As expected from Table 2 version 4 performs best. These performance numbers include the time required to load the data from the SSD into memory which, on average, required 2.0 minutes. Hence, when ignoring the loading of the data, we observe a factor 1.9 speed-up when using two CPU cores, a factor 3.6 speed-up when using four CPU cores, and a factor 6.9 speed-up when using eight CPU cores compared to running on a single core considering version 4 of the algorithm.

7.2 Second order CPA

We also implemented the various approaches from Section 5. The right plot of Fig. 4 reports the performance results for the fastest variants of version 7, version 8, and version 9. Since all the data-types used are 8-byte double-precision floating points these $tn = 2^9 \cdot 10^7$ values require over 38 GB of data: too big to load entirely into memory. Since a single time sample, consisting of 10^7 8-byte values, does fit into memory (this is just over 76 MB) we split the data across the time samples (as outlined in Section 6.1) and load them in batches of $s = 64$ time samples (which corresponds to batches of 4.9 GB). Extracting the correct s time samples of the power measurement file and loading this into memory requires 58 seconds per batch.

As expected from counting the number of required arithmetic instructions, version 8a performs the best among the considered variants which compute second order CPA. However, version 9b is considerably slower compared to the other two versions. Due to the relatively small windows size used ($w = 4$), version 9b computes a factor of 1.13 more multiplications compared to version 7b. However, the number of additions required in version 9b is reduced by a factor 1.60 compared to version 7b. Note that version 9b needs much more memory accesses in the innermost loop when computing the value of s_{10}: three memory loads are required at every iteration. We have conducted experiments which confirm that these increased number of (random) memory accesses explain the performance difference observed in practice.

7.3 Combining using Dunlap's approach

Figure 5 displays experimental results when applying the different combining approaches. For this experiment we used a (leaky) power measurement where $n = 10^5$. We varied the number of groups of traces (chunks) where we evenly divided the number of traces as $s_i = m/n$ for all i. Figure 5 shows the (estimated) correlation coefficient when using all n traces computed using Eq. (8), Eq. (7), and Eq. (6), respectively. In all three settings, as expected, more variation is observed as the number of chunks increases. The error percentage compared to the correct correlation coefficient is also shown in Figure 5; in all three settings the error is well below one percent (i.e. insignificant). Note that the values obtained when using Eq. (6), which is not an approximation, still exhibit a small error (of less than 0.2 percent). This can be explained by the propagation of errors due to the finite precision which is used. Hence, in all three settings this allows us to split the number of traces in 200 chunks (each only holding 500 traces), compute the correlation coefficient on these smaller datasets and combine them such the result has an error less than 0.3 percent.

8 Conclusions and Future Work

Computing the Pearson product-moment correlation coefficient in the setting of (higher-order) correlation power attacks allows one to perform a number of precomputations which are not available in other (non-cryptologic) research areas. We have outlined different techniques to reduce the number of arithmetic operations and enable the attacker to add new traces incrementally without much computational overhead. Specifically, we have studied second-order attacks and demonstrated how they can be extended to the setting of adding traces *on-the-fly*. Furthermore, we have shown how one can compute on large datasets using commonly available computer architectures equipped with a modest amount of memory. In this latter setting we have used techniques from Dunlap [9] to combine the correlation coefficient from different small sets into the (estimate of the) correlation coefficient of the combination of these sets. Our benchmark and performance results show that the implementation of these techniques behave as expected and give a significant speed-up (both in performance and memory consumption).

A similar analysis can be performed on higher order correlation power attacks. An interesting exercise would be to derive incremental formulas for these higher order attacks, comparable to Eq. (4) for first order or Eq. (5) for second order, allowing to add traces on-the-fly. Similarly, it would be interesting to study the usage of different correlation coefficient functions when computing (higher order) CPA and applying the time-memory trade-off paradigm. Examples include Spearman's rank correlation coefficient [27] or Kendall's rank correlation coefficient [12] (e.g. as applied in [2]).

References

1. T. Bartkewitz and K. Lemke-Rust. A high-performance implementation of differential power analysis on graphics cards. In E. Prouff, editor, *CARDIS 2011*, volume 7079 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2011.
2. L. Batina, B. Gierlichs, and K. Lemke-Rust. Comparative evaluation of rank correlation based DPA on an AES prototype chip. In T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, editors, *ISC 2008*, volume 5222 of *LNCS*, pages 341–354, Taipei, Taiwan, Sept. 15–18, 2008. Springer, Berlin, Germany.
3. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-order threshold implementations. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 326–343, Kaoshiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer, Berlin, Germany.
4. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient AES threshold implementation. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 267–284, Marrakesh, Morocco, May 28–30, 2014. Springer, Berlin, Germany.
5. E. Brier, C. Clavier, and F. Olivier. Optimal statistical power analysis. Cryptology ePrint Archive, Report 2003/152, 2003. `http://eprint.iacr.org/2003/152`.
6. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29, Cambridge, Massachusetts, USA, Aug. 11–13, 2004. Springer, Berlin, Germany.
7. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer, Berlin, Germany.
8. J. Daemen and V. Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer, 2002.
9. J. W. Dunlap. Combinative properties of correlation coefficients. *The Journal of Experimental Education*, 5(3):286–288, 1937.
10. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Ç. Koç and C. Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172, Worcester, Massachusetts, USA, Aug. 12–13, 1999. Springer, Berlin, Germany.
11. M. Joye, P. Paillier, and B. Schoenmakers. On second-order differential power analysis. In J. R. Rao and B. Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 293–308, Edinburgh, UK, Aug. 29 – Sept. 1, 2005. Springer, Berlin, Germany.
12. M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.
13. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer, Berlin, Germany.
14. S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2007.
15. S. Mangard, E. Oswald, and F. Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
16. L. Mather, E. Oswald, and C. Whitnall. Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 243–261, Kaoshiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer, Berlin, Germany.
17. T. S. Messerges. Using second-order power analysis to attack DPA resistant software. In Ç. K. Koç and C. Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 238–251, Worcester, Massachusetts, USA, Aug. 17–18, 2000. Springer, Berlin, Germany.
18. A. Moradi, M. Kasper, and C. Paar. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the xilinx virtex-4 and virtex-5 bitstream encryption mechanism. In O. Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 1–18, San Francisco, CA, USA, Feb. 27 – Mar. 2, 2012. Springer, Berlin, Germany.
19. A. Moradi and O. Mischke. On the simplicity of converting leakages from multivariate to univariate - (case study of a glitch-resistant masking scheme). In G. Bertoni and J.-S. Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 1–20, Santa Barbara, California, US, Aug. 20–23, 2013. Springer, Berlin, Germany.
20. A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-enhanced power analysis collision attack. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 125–139, Santa Barbara, California, USA, Aug. 17–20, 2010. Springer, Berlin, Germany.
21. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of AES. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

22. F. Mueller. A library implementation of POSIX threads under UNIX. In *USENIX Winter*, pages 29–42, 1993.
23. K. Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58(347-352):240–242, 1895.
24. A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling. Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, Apr. 2011.
25. E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *Computers, IEEE Transactions on*, 58(6):799–811, June 2009.
26. K. Schramm and C. Paar. Higher order masking of the AES. In D. Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225, San Jose, CA, USA, Feb. 13–17, 2006. Springer, Berlin, Germany.
27. C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.
28. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The world is not enough: Another look on second-order DPA. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 112–129, Singapore, Dec. 5–9, 2010. Springer, Berlin, Germany.
29. J. Waddle and D. Wagner. Towards efficient second-order power analysis. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 1–15, Cambridge, Massachusetts, USA, Aug. 11–13, 2004. Springer, Berlin, Germany.