# FastCPA: Efficient Correlation Power Analysis Computation with a Large Number of Traces

Quentin L. Meunier

# FastCPA: Efficient Correlation Power Analysis Computation with a Large Number of Traces

Quentin L. Meunier
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France
quentin.meunier@lip6.fr

## ABSTRACT

Cryptographic algorithm implementations need to be secured against side-channel attacks. Correlation Power Analysis (CPA) is an efficient technique for recovering secret key bytes of a cryptographic algorithm implementation by analyzing the power traces of its execution. Although CPA usually does not require a lot of traces to recover secret key bytes, it is no longer true in a noisy environment, for which the required number of traces can be very high. Computation time can then become a major concern for performing this attack and assessing the robustness of an implementation against it.

This article introduces FastCPA, which is a correlation computation targeting the same goal as regular CPA, but based on power consumption vectors indexed by plaintext values. The main advantage of FastCPA is its fast execution time compared to the regular CPA computation, especially when the number of traces is high: for 100,000 traces, the speedup factor varies from 70 to almost 200 depending on the number of samples.

An analysis of FastCPA accuracy is made, based on the number of correct key bytes found with an increasing noise. This analysis shows that FastCPA performs similarly as the regular CPA for a high number of traces. The minimum required number of traces to get the correct key guess is also computed for 100,000 noisy traces and shows that FastCPA obtains similar results to those of regular CPA. Finally, although FastCPA is more sensitive to plaintext values than the regular CPA, it is shown that this aspect can be neglected for a high number of traces.

## KEYWORDS

Correlation Power Analysis, Side-Channel Attacks, Performance, Efficient Computation

**ACM Reference Format:**
Quentin L. Meunier. 2019. FastCPA: Efficient Correlation Power Analysis Computation with a Large Number of Traces. In *Sixth Workshop on Cryptography and Security in Computing Systems (CS2 '19), January 21, 2019, Valencia, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3304080.3304082


*CS2 '19, January 21, 2019, Valencia, Spain*

ACM ISBN 978-1-4503-6182-8/19/01…$15.00
https://doi.org/10.1145/3304080.3304082

## 1 INTRODUCTION

Cryptographic algorithm implementations need to be secured against side-channel attacks. Since the first differential power analysis (DPA) on the DES by Kocher *et. al* [8], a lot of techniques based on side-channels information have been developed and enhanced for recovering secrets or assessing security against leakage: multi-bit DPA [1, 11], correlation power analysis (CPA) [2, 10], partitioning power analysis [9], mutual information analysis [5, 12] and statistical tests, especially specific and non specific Welch's T-Tests [3, 14]. These attacks and measures are carefully taken into account by system designers, especially for embedded systems, for which an attacker can measure the power consumption and the electromagnetic emanations, which are two of the main physical quantities used for non invasive attacks.

This article focuses on CPA, as it is an efficient technique for recovering a key given a simple power consumption model, and it usually requires a number of traces smaller by an order of magnitude than a DPA for recovering a key byte. If for an unprotected implementation running on a device without noise, a hundred of traces are usually sufficient to recover the key, protected implementations can require a huge amount of traces to assess their protection w.r.t. a type of attacks. Besides, real world devices usually contain much noise, and an attack may work only if the number of traces is very large. Finally, to assess the protection of a secure device against certain types of attacks, the designers need to consider a large amount of traces, since it corresponds to the worst case scenario, and resisting to an attack using 1,000 traces does not mean resisting to an attack using 100,000 traces. For these reasons, some works now consider the resistance of devices against up to a million of traces or more [14]. Therefore, having an efficient way for evaluating all types of attacks and tests is critical, as illustrated in [15], which highlights the high computational demand of the CPA and proposes a parallel implementation for a CPA at any order. Besides, several works have brought significant improvements in the T-Test computation complexity [4, 13, 14].

The basics of CPA are described in Section 2, and related work is discussed in section 3. We then introduce FastCPA, which is an efficient correlation coefficient computation allowing to recover secret key bytes, and describe the approach in Section 4. We show that FastCPA allows to significantly reduce the computation time of the CPA, especially when the number of traces is large (typically more than 1,000) in Section 5. We also show that this modification gives similar results in terms of detection accuracy. Finally, we conclude in Section 6.

## 2 CPA BASICS

CPA was first introduced by Brier *et. al* in [2]. It is based on a power consumption model of the running device at some point in time,

which must depend on certain secret bits, and on some input bits which change for each trace (namely the plaintext). For the AES, the time instant mostly used for performing the attack is after the first SBox, because it is the place where the linearity between the plaintext and the key is broken and with the finest granularity, *i.e.* where key bytes are not yet mixed together, allowing for few key values hypotheses. Similarly, the attack can be performed before the last SBox and target the last round key. The most widely used consumption models are the Hamming distance between two relevant values (e.g. stored consecutively in the same register), or simply the Hamming weight of a particular value. Then, this model is correlated with actual power consumption measures using the empirical Pearson's correlation coefficient. This empirical correlation coefficient between two samples $x_i$ and $y_i$ ($1 \leq i \leq n$) is given by:

$$\hat{r} = \frac{\hat{\sigma}_{X,Y}}{\hat{\sigma}_X \hat{\sigma}_Y} \tag{1}$$

with

$$\hat{\sigma}_{X,Y} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y}) \tag{2}$$

$$\hat{\sigma}_X = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}, \hat{\sigma}_Y = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2} \tag{3}$$

where $\overline{x}$ and $\overline{y}$ are the empirical means of $x_i$ and $y_i$.

In the following, we focus on one key byte as the attack tries to recover key bytes independently. More specifically, given $N$ power traces of length $L$, we note by $t_{n,i}$ the consumption value of point $i$ in trace $n$ (with $1 \leq n \leq N, 1 \leq i \leq L$). For $K$ possible subkeys (typically $K = 256$), we note $h_{n,k}$ the power estimation in trace $n$ (depending on the plaintext), assuming the subkey is $k$ ($0 \leq k < K$). With these data, we can see how well our model and measurements match for each guess $k$ at each time instant $i$, by calculating:

$$r_{k,i} = \frac{\sum_{n=1}^{N}(h_{n,k} - \overline{h_k})(t_{n,i} - \overline{t_i})}{\sqrt{\sum_{n=1}^{N}(h_{n,k} - \overline{h_k})^2 \sum_{n=1}^{N}(t_{n,i} - \overline{t_i})^2}} \tag{4}$$

in which $\overline{h_k}$ and $\overline{t_i}$ are respectively the average values of the modeled power consumption and the measured power consumption at instant $i$.

We can show that $r_{i,k}$ can be equivalently computed with the following formula, which is more efficient because it allows for online computation (*i.e.* the data set needs to be traversed only once):

$$r_{k,i} = \frac{N\sum_{n=1}^{N} h_{n,k}t_{n,i} - (\sum_{n=1}^{N} h_{n,k})(\sum_{n=1}^{N} t_{n,i})}{\sqrt{((\sum_{n=1}^{N} h_{n,k})^2 - N\sum_{n=1}^{N} h_{n,k}^2)((\sum_{n=1}^{N} t_{n,i})^2 - N\sum_{n=1}^{N} t_{n,i}^2)}} \tag{5}$$

Then, by taking the maximum of the $r_{k,i}$ over all the values for $i$ and $k$, we can deduce which key hypothesis is the most probable (the maximum over $i$ is required since we do not know *a priori* at which instant in the trace our power model applies).

An algorithm for computing such a correlation coefficient based on Equation 5 is shown in Algorithm 1. In particular, we can note that the complexity of this algorithm is proportional to $N \times K \times L$, what can be somewhat prohibitive: an AES round typically is 6,000 samples (using 4 samples per cycle), what makes more than 153 billions iterations of the innermost loop for 100,000 traces!

**Algorithm 1** Algorithm for computing the correlation coefficient

**Require:** N traces with L points of measure, a power model $h_{n,k}$ for each key hypothesis $k$ and trace $n$
**Ensure:** The key byte value with highest correlation w.r.t. the power consumption model is returned

```
for all i ∈ [1; L] do
    sum_t_i ← 0
    sum_t_i^2 ← 0
end for
for all k ∈ [0; K − 1] do
    sum_h_k ← 0
    sum_h_k^2 ← 0
    for all i ∈ [1; L] do
        sum_ht_{k,i} ← 0
    end for
end for
for all n ∈ [1; N] do
    for all i ∈ [1; L] do
        sum_t_i ← sum_t_i + t_{n,i}
        sum_t_i^2 ← sum_t_i^2 + t_{n,i}^2
    end for
    for all k ∈ [0; K − 1] do
        sum_h_k ← sum_h_k + h_{n,k}
        sum_h_k^2 ← sum_h_k^2 + h_{n,k}^2
        for all i ∈ [1; L] do
            sum_ht_{k,i} ← sum_ht_{k,i} + t_{n,i} · h_{n,k}
        end for
    end for
end for
MaxCorrCoeff ← 0
MaxKHyp ← 0
for all i ∈ [1; L] do
    for all k ∈ [0; K − 1] do
        CorrCoeff ← (N·sum_ht_{k,i} − sum_h_k·sum_t_i) / sqrt((sum_h_k)^2 − N·sum_h_k^2)((sum_t_i)^2 − N·sum_t_i^2))
        if |CorrCoeff| >MaxCorrCoeff then
            MaxCorrCoeff ← |CorrCoeff|
            MaxKHyp ← k
        end if
    end for
end for
return MaxKHyp
```

In opposition, the method proposed in this article requires less than 1 billion iterations of the innermost loop for the same configuration.

## 3 RELATED WORKS

Since the first works on CPA, the focus has first been on the method accuracy, and recently moved on the computation effectiveness.

In [9], the authors try to unify the concepts of DPA, multi-bit DPA and CPA *via* the concept of Partitioning Power Analysis (PPA). The article defines classes in which all the traces are partitioned, depending on their Hamming weight at a given moment in the computation. The correlation is then computed on the classes using different weights for the different classes. The authors show that CPA is a particular case of PPA. Although the work presented in this article uses the same principle of classes as well, it is not a particular case of PPA since the classes are not based on Hamming weights but directly on values. Besides, [9] focuses on key recovery accuracy and does not consider computation time.

The work described in [7] presents an improvement of CPA which consists in selecting a biased subset of the power traces containing more information than the average traces. The selection process tries to maximize the signal to noise ratio, by selecting the traces which are located at the extremities of the empirical probability density function. This article does not focus on execution time, and in particular does not present a computational analysis in terms of execution time: overhead linked to the selection process, and time saved thanks to the reduction of traces.

More recently, [14] presented a very effective way to compute the T-Test requiring only a single pass on the set of traces, allowing to compute regularly the T-Test value and to potentially stop before all the traces have been processed. The method uses the higher order central moments and is stable at least up to the fifth order. However, the proposed method is not applicable to CPA.

The method presented in [13] also targets an effective computation of the T-Test by representing the set of traces at each instant by a vector containing, for each possible measured value, the number of times that the corresponding value has been measured. This technique allows to decouple the estimation of measurements distributions and the computation of the statistical moments, which thus becomes independent from the number of traces. The work presented in this article also uses the idea to gather values in a vector, this time not indexed by the measured value, but by the plaintext value. To the best of our knowledge, the method proposed in this article has not been proposed in prior works.

## 4 FASTCPA

This Section first presents our proposed algorithm for performing the correlation power analysis, and then we expose the rationale of the proposed approach compared to the original CPA.

### 4.1 Algorithm

Similarly to the original CPA, our proposed method for performing the correlation power analysis on an attacked key byte targets an intermediate value in the computation. The method is based on the idea to create, at each point in time, a vector of consumption values indexed by the plaintext byte value. The $i^{th}$ element of such a vector thus represents the average measured consumption for all runs where the plaintext value was $i$ at a given point in time. During the first phase of the algorithm, these vectors are created in a single pass, requiring $N \times L$ iterations.

In a second part, these vectors are correlated, for each point in time, with vectors of identical length containing a consumption model, for all key hypotheses. These consumption model vectors are also indexed by the plaintext value, and contain typically the Hamming weight of the targeted internal value. This second phase takes $L \times K^2$ iterations.

The proposed algorithm is shown in Algorithm 2. In this algorithm, the `CompCC` function calculates the correlation coefficient between the two vectors and is given in Algorithm 3.

**Algorithm 2** Proposed FastCPA algorithm

**Require:** N traces with L measure points, a pre-computed vector of the power consumption model $h_k$ for each key hypothesis $k$, with the associated means and variances. We typically have $h_k[\text{pt}] = h_{k,pt} = \text{HW}(\text{SBox}[k \oplus \text{pt}])$.
**Ensure:** The key byte value with highest correlation w.r.t. the power consumption model is returned

```
for all pt ∈ [0; K − 1] do
    for all i ∈ [1; L] do
        MeanPower_{i,pt} = 0;
        NumValues_{i,pt} = 0;
    end for
end for
for all n ∈ [1; N] do
    pt ← plaintext[n];
    for all i ∈ [1; L] do
        NumValues_{i,pt} ← NumValues_{i,pt} + 1
        MeanPower_{i,pt} ← MeanPower_{i,pt} + (t_{n,i} − MeanPower_{i,pt}) / NumValues_{i,pt}
    end for
end for
MaxCorrCoeff ← 0
MaxKHyp ← 0
for all i ∈ [1; L] do
    μ_0, σ_0^2 ← ComputeMeanAndVar(MeanPower_i)
    for all k ∈ [0; K − 1] do
        ▷ μ_1 and σ_1^2 are pre-computed values corresponding
        ▷ respectively to the mean and the variance of h_k
        ▷ The function CompCC computes the correlation coef-
        ▷ ficient between the two vectors MeanPower_i and h_k
        CorrCoeff ← CompCC(MeanPower_i, μ_0, σ_0^2, h_k, μ_1, σ_1^2)
        if |CorrCoeff| >MaxCorrCoeff then
            MaxCorrCoeff ← |CorrCoeff|
            MaxKHyp ← k
        end if
    end for
end for
return MaxKHyp
```

### 4.2 Rationale

Compared to the original correlation coefficient computation, this method puts the same weight on each of the plaintext possible value in the correlation computation, and correlates only K points. For example, if we consider a 2-bit plaintext for simplicity and if we have 1,000 traces for plaintext value "0", 100 for plaintext values "1"

**Algorithm 3** Function CompCC

**Require:** u and v two vectors of K values, with their respective means $\mu_0, \mu_1$ and their respective variances $\sigma_0^2, \sigma_1^2$
**Ensure:** The empirical correlation coefficient between the two vectors is returned
$\mu_{01} \leftarrow 0$
**for all** $k \in [0; K-1]$ **do**
$\mu_{01} \leftarrow \mu_{01} + \frac{u[k]v[k]-\mu_{01}}{k+1}$
**end for**
**return** $\frac{\mu_{01}-\mu_0\mu_1}{\sqrt{\sigma_0^2\sigma_1^2}}$

and "2", and only 5 for plaintext value "3", all four plaintext values consumptions are weighted the same and count as much in the proposed correlation computation. Indeed, it is not obvious that each of the 1,000 power consumption values for the plaintext value "0" should count as much as each of the 5 values for plaintext value "3": intuitively, we feel that this 5 values contain more information than 1/200 of all measured values. Yet, putting the same weight is still not ideal because of the variability – having less values means less confidence in the average – but we can see that the question of the best way of computing the correlation is not trivial. We can also notice that the quantities computed in the case where the number of traces is identical for each plaintext value are different from those of the original CPA. This shows the need to assess experimentally the efficiency of the proposed computation compared to the original one.

More formally, if we define $T_i$ the set of measures at instant $i$, $T_i^v$ the restriction of $T_i$ for traces in which the plaintext value is $v$, and $\eta_{k,v}$ the power consumption model for key guess $k$ and plaintext value $v$, the formula for the computed correlation coefficient is (assuming plaintext values are in $[0; K-1]$):

$$r_{k,i} = \frac{\sum_{v=0}^{K-1}(\eta_{k,v} - \overline{\eta_k})(\tau_{i,v} - \overline{\tau_i})}{\sqrt{\sum_{v=0}^{K-1}\left(\eta_{k,v} - \overline{\eta_k}\right)^2 \sum_{v=0}^{K-1}(\tau_{i,v} - \overline{\tau_i})^2}} \quad (6)$$

with:

$$\tau_{i,v} = \frac{1}{|T_i^v|}\sum_{t \in T_i^v} t,\ \overline{\tau_i} = \frac{1}{K}\sum_{v=0}^{K-1}\tau_{i,v},\ \overline{\eta_k} = \frac{1}{K}\sum_{v=0}^{K-1}\eta_{k,v} \quad (7)$$

We can note that $\overline{\tau_i}$ is not the average power consumption measured at instant $i$, but the average of the averages for all plaintext values.

## 5 EXPERIMENTAL EVALUATION

This Section evaluates FastCPA on a standard AES implementation compared to the traditional correlation coefficient used for CPA (Ref. CPA) w.r.t. two criteria: the execution time, and the accuracy of both methods for several configurations. For all experiments and configurations, the traces were recorded by the ChipWhispererPro (R) platform [6] with an Xmega target board.

### 5.1 Execution Time

We measured the execution times of both methods for a CPA targeting a key byte after the first SBox, for the configurations listed in Table 1.

**Table 1: Parameters of the CPA attack on an AES key byte for execution time measurements**

| | |
|---|---|
| Number of samples (L) | { 1,000; 3,000; 6,000; 10,000 } |
| Number of traces (N) | { 2,000; 5,000; 10,000; 20,000; 50,000; 100,000 } |
| Number of key values (K) | { 256 } |

The computations, implemented as a single-threaded program, were run on an Intel (R) Xeon E5-2637 v2 processor at 3.50 GHz, and the code for both versions has been compiled with gcc 4.8 and optimisation level O2. The results are shown in Figure 1.

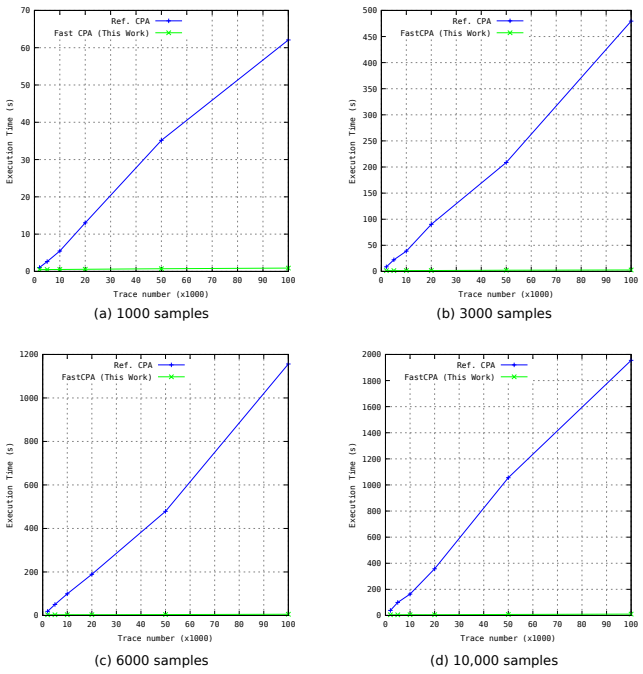


**Figure 1: Execution time results for both methods, for traces containing 1,000 samples (a), 3,000 samples (b), 6,000 samples (c), 10,000 samples (d), for a varying number of traces.**

We can see that the FastCPA computation method is way faster than the original CPA computation, achieving speedup factors from ×2 (2,000 traces of 1,000 samples) to ×191 (100,000 traces of 10,000 samples).

To allow for more readable results of FastCPA, we provide Table 2 which contains the execution times of this proposed method.

As expected, the larger the configuration is in terms of number of samples and traces, the greater speedup FastCPA exhibits compared to the reference CPA. We can note the that FastCPA computation

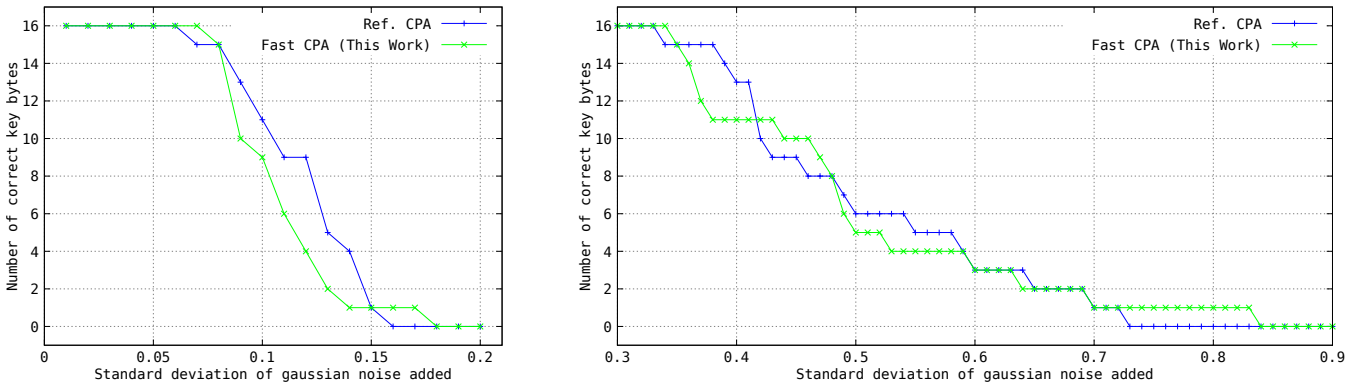


**Figure 2: Number of correct key bytes for both methods, with a varying gaussian noise added, for configurations 1 (a) and 2 (b)**

**Table 2: FastCPA Execution Time (in seconds)**

| | # Traces | | | |
|---|---|---|---|---|
| **# Samples** | **1,000** | **3,000** | **6,000** | **10,000** |
| **2,000** | 0.49 | 1.48 | 3.16 | 5.32 |
| **5,000** | 0.50 | 1.52 | 3.16 | 5.29 |
| **10,000** | 0.51 | 1.58 | 3.48 | 5.62 |
| **20,000** | 0.57 | 1.70 | 3.50 | 6.00 |
| **50,000** | 0.72 | 2.29 | 4.16 | 7.54 |
| **100,000** | 0.90 | 2.72 | 5.48 | 10.23 |

time seems less sensitive to the number of traces than to the number of samples: when the former is multiplied by 10, the execution time is roughly multiplied by 2, whereas when the latter is multiplied by 10, the execution time is roughly multiplied by 10 as well.

## 5.2 Method Accuracy

To evaluate the method accuracy w.r.t. the reference CPA, we cannot just compare the values of the correlation coefficients of both methods, as the coefficients will be higher with FastCPA due to the fact that the number of points is to correlate is smaller (256 vs. number of traces). Thus, we compare the number of correct key guesses as the noise increases, for a recorded set of traces and for both methods. The noise added is a centered gaussian noise.

Two configurations are considered for this experiment:

- Configuration 1: 5,000 traces comprising 3,000 samples, with a Gaussian noise added ranging from $\sigma = 0$ (16 correct key bytes for both methods) to $\sigma = 0.2$ (16 incorrect key bytes for both methods), with steps of 0.01;
- Configuration 2: 100,000 traces comprising 3,000 samples, with a Gaussian noise added ranging from $\sigma = 0.3$ (16 correct key bytes for both methods) to $\sigma = 0.90$ (16 incorrect key bytes for both methods), with steps of 0.01;

The values of $\sigma$ should be considered knowing that all the recorded samples values are in the interval [-0.7; 0.1].

The goal of configuration 1 is to assess that even with a relatively low number of traces, the proposed method performs decently. At the opposite, configuration 2 corresponds to the case where more traces are required to perform an attack because of the presence of noise, and therefore where more computation is required.

The results for configuration 1 are presented in Figure 2(a). We can see that for 5,000 traces, the reference CPA seems to perform a little better, since the drop in the number of key bytes correctly found happens for a slightly higher noise.

However, the results for configuration 2 with 100,000 traces indicate that while being a lot faster, the precision of FastCPA is very similar to the reference CPA, as shown on Figure 2(b). We believe that this experimentation illustrates the situations where FastCPA shows all of its usefulness.

To strengthen the confidence in FastCPA accuracy, we also measured for each byte the minimum number of traces required to get the correct key value (for any higher number of traces), and compared it for both methods. Obviously, for non noisy traces, the regular CPA performs better since it can get the correct key in 30 to 100 traces, whereas a minimum number of traces is required for FastCPA to get it (see section 5.3). Thus, we performed this measure for an arbitrary standard deviation noise value of 0.3, since all key bytes are still correctly found for both methods for such a noise. The results are shown in Figure 3.

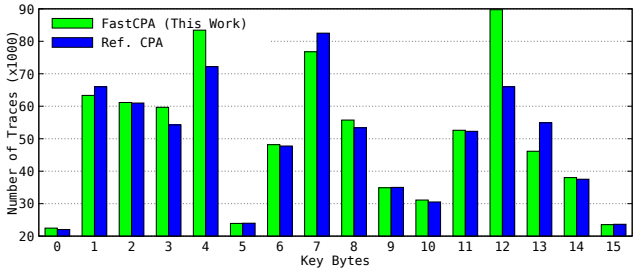


**Figure 3: Minimum number of traces required for getting the correct key guess for any greater number of traces, for a gaussian noise with standard deviation $\sigma = 0.3$. The lower the better.**

We can see that the results for both methods do not differ significantly, except for key byte 12 for which FastCPA requires almost 90K traces, while the reference CPA only requires 68K traces.

### 5.3 Plain Text Sensitivity

The proposed approach is more sensitive to the plaintext values in the sense that the correlation given by the function `CompCC` will be low until all (or almost) of the plaintext possible values have been reached by at least one run. This effect is visible on the Figure 4 which shows the correlation coefficients obtained for one key byte[1] for a number of traces varying between 1 and 5,000.

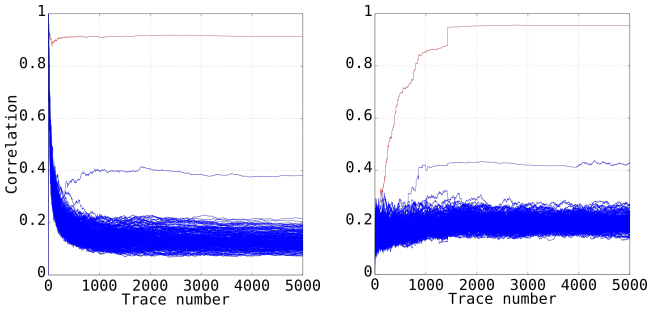


**Figure 4: Plain Text Sensitivity: correlations obtained for one key byte for both methods for each key guess, for a number of traces varying between 1 and 5,000. The red line corresponds to the correct key guess, the blue lines to bad key guesses.**

There are several ways to take this drawback into account. First, the traces can be ordered so that plain values do not appear twice as long as all of them have not been encountered. Second, the `CompCC` function can be adapted to only consider the points of the vector for which at least one run had the corresponding plaintext. This allows to suppress this effect and gives the curve a similar aspect to the one of the original correlation. However, this is not necessarily desirable since this lengthen the compute time of the function, and after a sufficient number of traces, the corresponding code modification becomes useless. Finally, table 3 shows the probability to not have reached all plaintext values, supposing uniformly and independent plaintexts, after some number of traces for 256 possible plaintext values.

**Table 3: Probability of not reaching all 256 plaintext values after a varying number of traces**

| # Traces | 500 | 1,000 | 2,000 | 3,000 | 4,000 |
|---|---|---|---|---|---|
| **Probability** | 100 % | 99.5 % | 9.7 % | 0.2 % | 0.004 % |

We can see that after 4,000 runs, the probability of not reaching all plaintext possible values is less than 0.005 %, suggesting that we can neglect this aspect for a high number of traces.

[1]The curves shapes are representative of all other key bytes

## 6 CONCLUSION

In this article, we proposed a new method, called FastCPA to perform a correlation power analysis in order to recover secret key bytes. This method consists in gathering measures by plaintext values and build a profile vector indexed by plaintext values, which is then correlated to a power consumption model vector. The proposed method achieves up to ×200 speedups compared to the original CPA computation, and is particularly efficient when the number of traces is high. The proposed method's accuracy to recover secret key bytes is comparable to the original CPA for two different metrics, namely the number of correct key bytes found for an increasing gaussian noise added to the traces, and the minimum number of traces required to get the correct key guess.

Future work includes the study of combining FastCPA with existing techniques for reducing the number of traces or for improving its accuracy. We also intend to adapt the correlation computation by giving different weights to the vector elemnts depending on the number of corresponding samples. Finally, we intend to make experiments with up to one million traces for both methods to assess the scalability of the proposed approach.

## REFERENCES

[1] Régis Bevan and Erik Knudsen. 2002. Ways to enhance differential power analysis. In *International Conference on Information Security and Cryptology*. Springer, 327–342.
[2] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 16–29.
[3] Jean-Sébasticn Coron, Paul Kocher, and David Naccache. 2000. Statistics and secret leakage. In *International Conference on Financial Cryptography*. Springer, 157–173.
[4] A Adam Ding, Cong Chen, and Thomas Eisenbarth. 2016. Simpler, faster, and more robust t-test based leakage detection. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 163–183.
[5] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. 2008. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 426–442.
[6] NewAE Technology Inc. [n. d.]. ChipWhisperer Pro Platform. https://newae.com, http://wiki.newae.com/CW1200_ChipWhisperer-Pro.
[7] Yongdae Kim, Takeshi Sugawara, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. 2010. Biasing power traces to improve correlation power analysis attacks. In *First International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE 2010)*. Citeseer, 77–80.
[8] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.
[9] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. 2006. A proposition for correlation power analysis enhancement. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 174–186.
[10] Rita Mayer-Sommer. 2000. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 78–92.
[11] Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. 2002. Examining smart-card security under the threat of power analysis attacks. *IEEE transactions on computers* 51, 5 (2002), 541–552.
[12] Emmanuel Prouff and Matthieu Rivain. 2009. Theoretical and practical aspects of mutual information based side channel analysis. In *International Conference on Applied Cryptography and Network Security*. Springer, 499–518.
[13] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. 2017. Fast Leakage Assessment. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 387–399.
[14] Tobias Schneider and Amir Moradi. 2015. Leakage assessment methodology. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 495–513.
[15] Tobias Schneider, Amir Moradi, and Tim Güneysu. 2015. Robust and One-Pass Parallel Computation of Correlation-Based Attacks at Arbitrary Order - Extended Version. Cryptology ePrint Archive, Report 2015/571. https://eprint.iacr.org/2015/571.