Selecting Time Samples for Multivariate DPA Attacks

Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede

KU Leuven Dept. Electrical Engineering-ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. Masking on the algorithm level, i.e. concealing all sensitive intermediate values with random data, is a popular countermeasure against DPA attacks. A properly implemented masking scheme forces an attacker to apply a higher-order DPA attack. Such attacks are known to require a number of traces growing exponentially in the attack order, and computational power growing combinatorially in the number of time samples that have to be exploited jointly. We present a novel technique to identify such tuples of time samples before key recovery, in black-box conditions and using only known inputs (or outputs). Attempting key recovery only once the tuples have been identified can reduce the computational complexity of the overall attack substantially, e.g. from months to days. Experimental results based on power traces of a masked software implementation of the AES confirm the effectiveness of our method and show exemplary speed-ups.

Keywords: Time sample selection, multivariate side-channel attack, masking, reverse-engineering.

1 Introduction

Side-channel attacks are used to break implementations of cryptographic algorithms in embedded devices. Since the introduction by Kocher [11] in the late nineties, they have been refined and a series of countermeasures have been designed to thwart them. A particularly popular countermeasure against Differential Power Analysis (DPA) attacks [12] is d-order masking [4, 7], since it enjoys a formal proof of security against higher-order DPA attacks [4, 15] of order d or less. d-order masking is based on splitting every sensitive intermediate value in $d + 1$ shares and we consider the case that they are manipulated at distinct times, as is typical for software implementations. $d + 1$-order DPA and $d + 1$-variate Mutual Information Analysis (MIA) attacks [5, 17] (from now on referred to as multivariate attacks together) allow to break d-order masked implementations by analyzing tuples of $d + 1$ time samples, corresponding to all shares of a masked sensitive variable, from each trace. However, multivariate attacks are significantly more difficult to mount than univariate attacks for two

reasons. First, attacks exploiting higher-order moments are exponentially more sensitive to noise as the masking order d increases [4, 19]. As a consequence, the number of traces required to mount a successful attack grows exponentially in d. Second, multivariate attacks need to search over $d+1$-tuples of time samples. The computational complexity of the attacks therefore grows combinatorially in the attack order $d+1$. Hence, secure implementations use a masking order d in combination with a suitable noise level to ensure that an attack will require a sufficiently large number of traces and a heavy amount of computation, such that the attack becomes impractical.

Related work. Most related works on non-profiled multivariate attacks start from the assumption that the time samples where the shares of the targeted, masked sensitive variable leak are known, and focus on the key recovery [5, 10, 15, 17, 18, 20, 23]. Few related works tackle the problem of identifying (tuples of) interesting time samples before key recovery, and they do so with heuristic approaches. Agrawal et al. [1] describe a method to identify tuples of time samples that requires a chosen input adversarial model and that can only exploit the leakage of single bits. Their method is tailored to Boolean masking and the measurements can not be re-used for key recovery, due to the way the inputs are chosen. Oswald et al. [16] essentially propose an exhaustive search over all $d+1$-tuples of time samples in a small time window that is selected based on an *educated guess*. The interpretation of *educated guess* is left to the practitioner. Note that the guess does not select tuples of time samples, but a window of time samples that has to be searched for a tuple exhaustively in combination with key recovery. This method can be applied with known inputs or outputs and, in principle, to any masking scheme. The approach suggested by Lemke and Paar [14] and Gierlichs et al. [5] is to examine the empirical variance of several power traces when the input data is kept constant, i.e. it requires a chosen input adversarial model. In an ideal case, the variance is then caused only by masking, and therefore time samples with high variance mostly correspond to time samples where the masks or masked variables are being processed. Note that also this method does not identify tuples of time samples but a set of samples that has to be searched for a tuple exhaustively in combination with key recovery. The measurements can not be re-used for key recovery and, in principle, the method can be applied to any masking scheme.

In summary, the *educated guess* of Oswald et al. is the only method described in the literature that can be applied in black-box conditions and with known inputs or outputs.

Contribution. We present a novel method for identifying interesting $d+1$-tuples of time samples before key recovery. It is not heuristic but systematic and ranks all possible $d+1$-tuples of time samples in a given window according to their dependency on, informally speaking, "typical attack targets". It does not provide a qualitative yes/no decision, but instead ranks tuples with respect to a meaningful metric such that there is a natural order in which to attack them. Our

technique can lead to a substantial improvement in the computational efficiency of multivariate attacks compared to exhaustive search over the same window of time samples, since it retains only a small fraction of all possible $d+1$-tuples for key recovery. The relative improvement depends on the size of the subkeys that are attacked. In absolute terms, the improvement becomes more pronounced with increasing attack order $d+1$, increasing size of the time window, and increasing number of traces.

Our approach is based on mutual information and is fully generic: it applies to attacks of any order $d+1$, including univariate attacks against unmasked implementations, it applies to all possible masking schemes, it requires only a known input or output scenario, it can traverse S-boxes, locate shares of the masked S-box output, and it does not require any restrictive assumptions on the device leakage behavior. In other words, our method does not require more restrictive assumptions than a generic MIA attack [6].

Paper organization. In Sect. 2 we introduce our notation, recall the basics of masking and discuss state-of-the-art multivariate attacks. In Sect. 3 we present our technique together with an analysis of how and why it works. We discuss its efficiency, impact, and possible refinements in Sect. 4. In Sect. 5 we present experimental results that validate our proposal and highlight some of its interesting properties. Section 6 concludes the paper.

2 Preliminaries

In this paper we consider only non-profiled, multivariate attacks. *Interesting* tuples are tuples of time samples that carry leakage of all shares of a masked variable that is a (possibly keyed) function of the plaintext.

2.1 Notation

Capital letters in bold face, e.g. $\mathbf{M}$, denote random variables. Lowercase letters, e.g. m, denote a specific value of $\mathbf{M}$, e.g. $\mathbf{M} = m$. $\mathbf{M}_i$ are mask bytes, $\mathbf{P}$ is a plaintext byte, $\mathbf{K}$ is a key byte, and `S-box` is a cryptographic S-box. $\mathbf{L}(t)$ is the random variable corresponding to the measured side-channel leakage at time t. $t_{\mathbf{M}}$ denotes the instant when the device is manipulating the random variable $\mathbf{M}$. $\mathbf{I}(\mathbf{A};\mathbf{B};\mathbf{C})$ denotes the multivariate mutual information between $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ [2, 5] and is computed as

$$\mathbf{I}(\mathbf{A};\mathbf{B};\mathbf{C}) = \mathbf{I}(\mathbf{A};\mathbf{B}) - \mathbf{I}(\mathbf{A};\mathbf{B}|\mathbf{C})\,. \tag{1}$$

Note that, if $\mathbf{A}$ and $\mathbf{B}$ are independent, $\mathbf{I}(\mathbf{A};\mathbf{B}) = 0$ and $\mathbf{I}(\mathbf{A};\mathbf{B};\mathbf{C}) \leq 0$.

2.2 Masking

Masking was introduced by Goubin and Patarin [7] and by Chari et al. [4] (together with a proof of security) as a sound approach to protect implementations

against first-order DPA attacks. In a d-order masked implementation, every sensitive variable $\mathbf{Z}$ is randomly split into $d+1$ shares $\mathbf{M}_1, \ldots, \mathbf{M}_d, \mathbf{V}$ satisfying

$$\mathbf{M}_1 \star \ldots \star \mathbf{M}_d \star \mathbf{V} = \mathbf{Z}\,, \tag{2}$$

where $\star$ is some suitable group operator.

The security of properly implemented masking schemes relies on the fact that even if the adversary manages to know any information about up to d shares out of $d+1$ (for example, via side-channel leakage), he cannot learn any information about the sensitive variable $\mathbf{Z}$.

Throughout the paper we assume that the shares $\mathbf{M}_1, \ldots, \mathbf{M}_d, \mathbf{V}$ are manipulated (and leak) separately at different time instants. Further, we assume that these time instants and the values of the shares are unknown to the adversary.

2.3 Multivariate attacks

Masked implementations of order d can in theory always be broken by $d+1$-variate attacks as originally proposed by Messerges [15] and Chari et al. [4]. They exploit the statistical dependence between the leakage of the $d+1$ shares and the sensitive variable $\mathbf{Z}$. There are essentially two different methods for performing multivariate attacks.

The first approach [4, 10, 15, 16, 18, 23] consists in reducing the problem to a univariate scenario by preprocessing each trace, and then running a first-order attack on the preprocessed traces. The preprocessing generates a new trace from all possible $d+1$-tuples of distinct time samples of the original trace, where for each tuple the $d+1$ time samples are combined with a so-called *combination function* (typically the absolute difference [15] or the centered product [18]). The second approach, proposed by Prouff and Rivain [17] and Gierlichs et al. [5], does not rely on a preprocessing step but directly uses multivariate MIA for the attack.

A major shortcoming of both methods is that they suffer from the effect known as "combinatorial explosion" and hence combinatorial time complexity in $d+1$. Both methods aim to recover subkeys while, at the same time, searching for a suitable $d+1$-tuple of time samples in the traces. In the first approach, the preprocessed traces are $\binom{L}{d+1}$ time samples long, where L is the trace length. These traces have to be processed for each hypothesis on the subkey. In the second approach, the distinguisher should be computed for each of the $\binom{L}{d+1}$ $d+1$-tuples, and for each hypothesis on the subkey.

Hence, it is very important to identify the interesting tuples (or to narrow down a window of time samples as much as possible) prior to key recovery in order to keep the computational complexity of a multivariate attack at a feasible level.

3 Identifying interesting tuples of time samples

In this section we explain how to identify interesting $d+1$-tuples of time samples prior to key recovery. Note that we focus our attention on this aspect and that

key recovery is not the primary focus of the paper. For clarity in the exposition, in what follows we assume a first-order Boolean masking scheme (two shares) and a noise-free scenario. The practical results presented in Sect. 5 are based on measured power traces.

3.1 Core idea

Let us consider a scenario with fixed plaintext, fixed key, and sensitive intermediate value $\mathbf{Z} = F_k(p)$, where F_k is some key-dependent function (for example, $F_k(p) = \texttt{S-box}(p \oplus k)$). The key observation is that the mutual information between the leakages at time instants corresponding to the manipulation of the mask $\mathbf{M}_1$ and the masked intermediate value $\mathbf{V} = \mathbf{M}_1 \oplus F_k(p)$ is non-zero. That is,

$$\mathbf{I}(\mathbf{L}(t_{\mathbf{M}_1}); \mathbf{L}(t_{\mathbf{V}})) > 0\,. \tag{3}$$

The interpretation is straightforward: leakage at $t_{\mathbf{V}}$ depends only on $\mathbf{V}$, which varies in function of only the mask $\mathbf{M}_1$ (since the plaintext and the key are fixed), and some information about the mask is leaked at $t_{\mathbf{M}_1}$. Hence, the information shared between leakage at $t_{\mathbf{M}_1}$ and $t_{\mathbf{V}}$ is non-zero. On the other hand, the information shared between leakage at two unrelated time samples t_0 and t_1 is zero

$$\mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1)) = 0 \tag{4}$$

because no relation exists between data handled at t_0 and at t_1. Thus, Eqs. (3) and (4) allow us to distinguish pairs of time samples that contain leakage of dependent variables (case of Eq. (3)) from those pairs that contain leakage of independent variables, that are irrelevant for the multivariate attack (case of Eq. (4)). Note that not all pairs of time samples that contain leakage of dependent variables carry some information about the key. For example, if the same mask is manipulated at t_0 and t_1, then $\mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1)) > 0$.

We stress that the value of $\mathbf{K}$ need not be known, and no hypothesis on it be made.

The general case. In the above example we required a fixed plaintext and thus a chosen plaintext scenario. We can relax this assumption and instead work with known (varying) plaintexts. Suppose that the device is manipulating the plaintext byte $\mathbf{P}$, the mask $\mathbf{M}_1$ and the masked intermediate value $\mathbf{V}$ such that $\mathbf{V} = \mathbf{M}_1 \oplus F_k(\mathbf{P})$ at time instants $t_{\mathbf{P}}$, $t_{\mathbf{M}_1}$ and $t_{\mathbf{V}}$, respectively. The natural extension of the core observation to known varying plaintexts is that $\mathbf{L}(t_{\mathbf{P}})$, $\mathbf{L}(t_{\mathbf{M}_1})$ and $\mathbf{L}(t_{\mathbf{V}})$ are not independent, and therefore the mutual information between them is non-zero

$$\mathbf{I}(\mathbf{L}(t_{\mathbf{M}_1}); \mathbf{L}(t_{\mathbf{V}}); \mathbf{L}(t_{\mathbf{P}})) \neq 0\,. \tag{5}$$

At three unrelated time samples t_0, t_1 and t_2, on the other hand, the mutual information is zero

$$\mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1); \mathbf{L}(t_2)) = 0\,. \tag{6}$$

The interpretation follows the same lines as in the particular case. Leakage at $t_{\mathbf{V}}$ depends only on $\mathbf{V}$, which now varies in function of the plaintext and the mask (since the key is fixed), and some information about the mask and the plaintext is leaked at $t_{\mathbf{M}_1}$ and $t_{\mathbf{P}}$, respectively. Thus, the information shared between $\mathbf{L}(t_{\mathbf{M}_1})$, $\mathbf{L}(t_{\mathbf{V}})$ and $\mathbf{L}(t_{\mathbf{P}})$ is non-zero.

Note that it is not necessary to search for $t_{\mathbf{P}}$, nor is it necessary for $t_{\mathbf{P}}$ to physically exist in the power traces. By assumption, the plaintext is known, so it is possible to substitute $\mathbf{L}(t_{\mathbf{P}})$ with the leakage of the known plaintext under some hypothesized leakage model $\tilde{\mathbf{L}}(\mathbf{P})$. This makes the analysis faster since one has to search only for a pair of time instants ($t_{\mathbf{M}_1}$ and $t_{\mathbf{V}}$) instead of searching for a triplet. The choice of $\tilde{\mathbf{L}}$ will be discussed in Sect. 3.3.

We can hence use Eqs. (5) and (6) to distinguish dependent triplets from independent triplets. In addition, and contrary to the particular case with fixed plaintext, all identified tuples are now interesting tuples and all relate to the *specific* plaintext byte $\mathbf{P}$. Most of them carry some information about the key and can be useful for a key recovery attack. The only possible type of tuple that will be identified as interesting although it does not carry some information about the key is the one corresponding to all shares of the specific, masked plaintext byte. We discuss this in more detail in Sect. 3.3.

3.2 Suggested workflow for multivariate attacks

The previous observations allow an attacker to identify interesting $d+1$-tuples of time samples prior to key recovery. Again, we use $d = 1$ in the explanation. The proposed workflow divides an attack in three phases:

Step 1. (Window selection) The adversary uses any available mean to narrow down the time window to analyze. For example, the adversary could select a small window based on an *educated guess* [16], if possible. Obviously, care has to be taken to not discard too many time samples since the window must contain at least one interesting $d+1$-tuple.

Step 2. (Tuple selection) The adversary estimates $\mathbf{I}(\mathbf{L}(t_1); \mathbf{L}(t_2); \tilde{\mathbf{L}}(\mathbf{P}))$ for all (t_1, t_2) with $t_1 > t_2$ in the remaining window, and keeps a list of pairs of time samples yielding negative mutual information with large absolute value.

Step 3. (Key recovery attack) The adversary performs the preferred strategy for a bivariate attack on traces consisting only of the pairs of time samples in the list. These traces consist of a few pairs of time samples, and hence the key recovery step is much faster.

3.3 Which tuples of time samples pop up?

The adversary has freedom to choose the hypothesized leakage model $\tilde{\mathbf{L}}$ for the plaintext. Depending on the choice of $\tilde{\mathbf{L}}$, different tuples of time samples will be identified. In this section we analyze two cases.

$\tilde{\mathbf{L}}$ is the identity function. When the adversary computes the mutual information between time samples and a plaintext byte, i.e. $\tilde{\mathbf{L}}(\mathbf{P}) = \mathbf{P}$, he will be able to identify all tuples corresponding to all shares of any (sensitive) variable of the form $\mathbf{Z} = F_k(\mathbf{P})$. In particular, the method is able to identify the shares $(\mathbf{M}_1, \mathbf{V})$ with $\mathbf{V} = \mathbf{P} \oplus \mathbf{M}_1$, $\mathbf{V} = \mathbf{P} \oplus \mathbf{K} \oplus \mathbf{M}_1$ and $\mathbf{V} = \texttt{S-box}(\mathbf{P} \oplus \mathbf{K}) \oplus \mathbf{M}_1$, since the key is fixed.

This result is useful, as it allows the attacker to locate both the masked variables *before* the S-box (masked plaintext and masked S-box input) as well as the masked variables *after* the S-box (masked S-box output). Note that it is irrelevant if the masks before and after the S-box are the same. If the mask does not change, the identified tuples of time samples will share one component.

$\tilde{\mathbf{L}}$ is an approximation of the device leakage behavior. If the attacker chooses $\tilde{\mathbf{L}}$ as an approximation of the leakage behavior $\mathbf{L}$, he will be able to identify all tuples of time samples corresponding to all shares of any (sensitive) variable of the form $\mathbf{Z} = F_k(\mathbf{P})$ appearing *before* the S-box (e.g. masked plaintext and masked S-box input). For a typical S-box, he will not be able to identify tuples of time samples corresponding to shares of sensitive variables *after* the S-box. The intuitive reasoning behind this is that knowledge of the distribution of the plaintext's *leakage* does not give sufficient information for guessing the distribution of the S-box output's leakage. The advantage of this choice, compared to the identity function, is the ease of estimation, see Sect. 4.1. Disadvantages are that one cannot locate shares of masked variables after the S-box and that one relies on an assumption about the device leakage behavior.

Note that we compute the mutual information according to Eq. (1), and not as

$$\begin{aligned} &\mathbf{I}((\mathbf{L}(t_0), \mathbf{L}(t_1)); \tilde{\mathbf{L}}(\mathbf{P})) = \\ &\qquad \mathbf{I}(\mathbf{L}(t_0); \tilde{\mathbf{L}}(\mathbf{P})) + \mathbf{I}(\mathbf{L}(t_1); \tilde{\mathbf{L}}(\mathbf{P})) - \mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1); \tilde{\mathbf{L}}(\mathbf{P}))\,, \end{aligned} \tag{7}$$

where the last of the three terms is in turn given by Eq. (1) [2]. The reasoning for this choice is straightforward. The first two terms of Eq. (7) capture first-order leakage of variables that depend on $\tilde{\mathbf{L}}(\mathbf{P})$, e.g. unmasked plaintext, unmasked S-box input and, depending on the choice of $\tilde{\mathbf{L}}$, unmasked S-box output. By assumption, the masking scheme is properly implemented and there is no first-order leakage of sensitive variables. Hence, the only first-order leakage that these terms could capture is that of the unmasked plaintext, which is of no use for our purpose. By omitting the two terms and using Eq. (1) we ensure that only interesting tuples yield non-zero mutual information.

Moreover, Eq. (1) allows us to target very specific tuples. For our interesting tuples it holds that $\mathbf{I}(\mathbf{L}(t_{\mathbf{M}_1}), \mathbf{L}(t_{\mathbf{V}})) = 0$ such that interesting tuples yield strictly negative mutual information, see (1).

4 Discussion

In this section we discuss several aspects of the proposed workflow for multivariate attacks, such as its efficiency, refinements and additional applications.

4.1 Efficiency analysis

We evaluate the efficiency of the proposed workflow with respect to the running time and the number of traces needed, and we compare these numbers to those of a "classical" multivariate MIA attack that uses exhaustive search instead of step 2. Although the proposed method is not limited to a particular multivariate attack technique for step 3, using multivariate MIA here allows us to draw important conclusions regarding the efficiency of the proposed workflow, since the numbers can be directly compared. In both cases we focus the attacks on the (masked) S-box output. According to the previous section, this choice implies that step 2 of the proposed workflow uses the identity function $\tilde{\mathbf{L}}(\mathbf{P}) = \mathbf{P}$. We analyze two different scenarios:

(a) Unknown leakage behavior. Step 3 of the proposed workflow and the "classical" MIA both use the identity leakage model, or possibly some truncated identity leakage model in case of a bijective S-box. The point here is that both step 3 and the "classical" MIA use the same leakage model.
(b) Known leakage behavior $\mathbf{L}$ equal to Hamming weight leakage. Step 3 of the proposed workflow and the "classical" MIA both use the Hamming weight leakage model.

Running time. We assume that after step 1 the traces are L time samples long and contain at least one tuple of time samples corresponding to all shares of the masked S-box output. We further assume that all attacks are provided with sufficiently many traces, i.e. there are no PDF estimation problems.

In scenario (a) the running time of the "classical" MIA attack is given by $\binom{L}{d+1} \times \alpha \times |K|$, where $\binom{L}{d+1}$ is the number of $d+1$-tuples of time samples to analyze, α is the time it takes to compute the MIA distinguisher for one $d+1$-tuple of time samples and one subkey hypothesis using the identity leakage model, and $|K|$ is the number of subkey hypotheses. In scenario (b) the running time of the "classical" MIA attack is $\binom{L}{d+1} \times \beta \times |K|$, where β is the time it takes to compute the MIA distinguisher for one $d+1$-tuple of time samples and one subkey hypothesis using the Hamming weight leakage model.

In scenario (a) the running time of step 2 of the proposed workflow is given by $\binom{L}{d+1} \times \alpha$, and the running time of step 3 is $|K| \times \alpha \times \gamma$, where γ is the number of $d+1$-tuples in the list of interesting tuples generated in step 2. We have that $\gamma \geq 1$ and typically γ is much smaller than L. The combined running time of steps 2 and 3 is $\binom{L}{d+1} \times \alpha + |K| \times \alpha \times \gamma$. In scenario (b) the running time of step 2 is again $\binom{L}{d+1} \times \alpha$ and the running time of step 3 is $|K| \times \beta \times \gamma$. The combined running time of both steps is $\binom{L}{d+1} \times \alpha + |K| \times \beta \times \gamma$. Note that

in both scenarios (a) and (b), the total running time of the proposed workflow is dominated by step 2. Table 1 summarizes these numbers and shows that the proposed workflow essentially runs $|K|$ times faster.

Table 1. Running time of MIA attacks using the proposed and the "classical" workflow.

	Proposed workflow	"Classical" MIA	Improvement factor
Scenario (a)	$\binom{L}{d+1} \times \alpha + \|K\| \times \alpha \times \gamma$ $\approx \binom{L}{d+1} \times \alpha$	$\binom{L}{d+1} \times \alpha \times \|K\|$	$\approx \|K\|$
Scenario (b)	$\binom{L}{d+1} \times \alpha + \|K\| \times \beta \times \gamma$ $\approx \binom{L}{d+1} \times \alpha$	$\binom{L}{d+1} \times \beta \times \|K\|$	$\approx \|K\| \times \beta/\alpha$

So far we have limited this analysis to attacks against a single subkey. For attacking multiple subkeys, it may be that only recovering the first subkey is hard and that the interesting tuples of time samples related to the other subkeys can be easily guessed once the tuple related to the first subkey has been found. But it may also be that recovering the other subkeys requires basically the same computation as recovering the first subkey. In either case, the improvement factor is essentially $|K|$. In the latter case, this improvement applies to recovering *each* subkey, which is not obvious since we express the improvement as a factor. Further, we note that the improvement factor is independent of the masking order d and the window size L. However, in absolute terms the running time improvement increases substantially with increasing attack order $d+1$, L and the number of traces. Finally, we point out that the analysis holds independently of the method used to estimate the mutual information, as long as we assume that all involved estimations of mutual information use the same method.

Number of traces needed. It is not straightforward to make a precise but general statement about the number of traces needed for our method to successfully locate interesting tuples. Many factors play a role. We make a brief assessment and describe two of the effects that have to be considered.

First, we consider an idealized scenario where steps 2 and 3 succeed as soon as the same precision for the estimations is achieved. In this case, in scenario (b) (Hamming weight leakage model), step 2 may require more traces to pinpoint the interesting $d+1$ tuples of time samples than step 3 to recover the key. This is due to the fact that, in the attack step, the estimation of $\mathbf{I}(\mathbf{L}(t_{\mathbf{V}}); \mathbf{L}(t_{\mathbf{M}}); \mathrm{HW}(\mathbf{Z}))$ with $\mathbf{Z} = \texttt{S-Box}(\mathbf{P} \oplus k)$ for a hypothesized k requires generally less traces than an equally precise estimation of $\mathbf{I}(\mathbf{L}(t_{\mathbf{V}}); \mathbf{L}(t_{\mathbf{M}}); \mathbf{P})$ in the tuple selection step. This is because of the different number of classes for $\mathrm{HW}(\mathbf{Z})$ and for $\mathbf{P}$. In the case of AES, there are 256 different possible values for $\mathbf{P}$, while there are only nine different possible values for $\mathrm{HW}(\mathbf{Z})$. Nevertheless, since step 2 requires a larger number of traces, these traces must be obtained and may be used in step

3. A "classical" $d+1$-variate MIA attack requires the same (smaller) number of traces as step 3.

In scenario (a) ((possibly truncated) identity leakage model) the previous effect is typically less pronounced and thus the difference in the number of traces required in each step is smaller. The same holds for the difference in the number of traces needed for step 2 and a "classical" $d+1$-variate MIA attack.

Second, the precision of the mutual information estimates required in step 3 to distinguish the correct key hypothesis from incorrect ones may not be the same as the precision required to distinguish an interesting tuple from a non-interesting one in step 2. The relation between these precisions can be almost arbitrary. However, it should typically hold that the precision required by an attack against the S-box output in step 3 is not higher than the precision required in step 2.

Summarizing, in scenario (a) the proposed workflow offers a running time improvement factor in the order of magnitude of $|K|$, possibly at the cost of an increased number of traces. In scenario (b) the proposed workflow requires more traces than a "classical" attack but still offers an interesting running time improvement factor. It offers a trade off. Whether the trade off is attractive depends on the ratio β/α in the running time improvement factor, and on how many more traces are required.

4.2 On the S-box

The fact that the method can distinguish all $d+1$ tuples corresponding to all shares of any (sensitive) variable of the form $\mathbf{Z} = F_k(\mathbf{P})$ can be used to traverse bijective S-boxes without making any hypothesis on the subkey. This is because the S-box input is a keyed permutation of the plaintext, and the S-box output is a permutation of the S-box input. Both permutations are transparent to mutual information when using the identity function $\tilde{\mathbf{L}}(\mathbf{P}) = \mathbf{P}$.

It is less obvious, nevertheless true, that the method also works in the case of non-injective S-boxes, as for instance in DES. The reasoning is similar to the above. The S-box input is a keyed permutation of the plaintext. The S-box output is not a permutation of the S-box input, but a non-injective function of it. Therefore, if we use the identity function and condition on the plaintext, the S-box can be traversed just like a bijective S-box, and interesting tuples of time samples after the S-box can be identified. Note that a non-injective S-box cannot be traversed from output to input in the same way.

4.3 Additional applications

The method described in this paper is fully generic and does not place any restrictive assumption on the specific targeted implementation. However, the method benefits from the available specificities of the implementation. For example, an adversary could mount the following strategy if he knows that the device's leakage behavior is close to the Hamming weight model. Using the Hamming weight model, the adversary first locates tuples corresponding to the S-box input to

narrow down the time window. Then, using the identity function, he searches in that window for the S-box output.

The adversary could also locate tuples corresponding to the S-box input of the next S-box lookup to further narrow down the time window.

Bit-tracing [9] is a technique used to track the time instants when a predictable variable is handled in the execution flow of an unknown implementation. This is a useful technique to reverse-engineer unknown implementations. The ideas in Section 3.1 can be exploited to track masked variables during the execution of an algorithm. Note that the fact that the proposed method can traverse S-boxes (by the arguments given in Sect. 4.2) can also lead to a significant speed-up in the bit-tracing process of masked implementations.

4.4 Estimation of mutual information

We note that any suitable method for estimating the mutual information or the required probability distributions, e.g. histograms [6], kernel density estimation [17], B-splines [21], statistical moments [13], parametric methods [17], and any similar metric, e.g. Kullback-Leibler divergence [22], Kolmogorov-Smirnov test [22], Cramér-von-Mises test [22], can be used.

Available knowledge about the device leakage behavior, e.g. close to the Hamming weight model, can be used to speed up the estimations. Here we do not refer to the choice of $\tilde{\mathbf{L}}$ but to the leakage variables.

4.5 Key recovery step

By construction, our method identifies tuples of time samples that correspond to all shares of a masked (sensitive) variable. It does so irrespective of the particular dependencies between each share and its side-channel leakage. Therefore, a generic multivariate MIA attack with (possibly truncated) identity leakage model appears to be most suited to exploit the unknown dependencies, in general. However, if standard assumptions approximate the leakage behavior good enough or the specific leakage behavior is known, the identified tuples can be exploited more efficiently with adapted multivariate MIA or higher-order DPA attacks.

The proposed method can identify interesting tuples that relate to a specific plaintext byte, but it cannot *per se* focus on interesting tuples that correspond to a *specific* function of that plaintext byte. As a consequence, the method will in general not discriminate between interesting tuples that correspond to all shares of the masked plaintext, the masked S-box input or the masked S-box output. Clearly, the latter is preferable for an attack. In our experiments we noted that enough interesting tuples corresponding to all shares of the masked S-box output appeared at the top of the ranked list.

5 Experiments

In this section we present experimental results of our method, insight on its computation and a performance evaluation. We note that all "numbers of traces" reported in this section cannot be generalized to other platforms and implementations.

5.1 Measurements

We use an 8-bit microcontroller of Atmel's AVR family in a smart card plastic body as platform for our experiments. The microcontroller runs a first-order Boolean masked implementation of AES-128 encryption that follows the lines of [8]. This concrete implementation uses six independent mask bytes for one encryption. Before the SubBytes operation, all state bytes are protected by the same mask $\mathbf{M}_0$. After the SubBytes operation, all state bytes are protected by the same mask $\mathbf{M}_1$. Before MixColumns, each column of the state is remasked with $\mathbf{M}_2, \ldots, \mathbf{M}_5$. After MixColumns, each column of the state is masked with $\mathbf{M}'_2, \ldots, \mathbf{M}'_5$ that depend on $\mathbf{M}_2, \ldots, \mathbf{M}_5$. Shiftrows does not affect the masking and after the next AddRoundKey operation, all state bytes are again protected by $\mathbf{M}_0$ due to the masked key schedule. Note that the six masks are re-used to protect all rounds. There are no additional countermeasures.

We obtained 50 000 power traces from encryptions of randomly chosen plaintexts with a fixed key and random masks. The card was clocked at 4MHz and we used a sampling frequency of 200MS/s.

5.2 Selection of a time window: step 1

To reduce the computational burden, we restricted the measurements to cover only the first 1.5 rounds of the encryption. This was done based on an educated guess on the SPA features present in the power traces. Then, we compressed the traces by integration to one point per clock cycle. As a result, each compressed trace comprises 800 points. The subsequent analyses were carried out on these compressed traces.

5.3 Computation of the method: step 2

To show the full potential of the method, we chose $\tilde{\mathbf{L}}$ to be the identity function. In what follows, $\mathbf{P}$ refers to the third plaintext byte, an arbitrary choice. We estimate densities with histograms (using nine bins for each dimension unless otherwise stated, because we expect a leakage behavior close to the Hamming weight/distance model) and we use $\hat{}$ to indicate estimates, e.g. $\hat{\mathbf{I}}$ is an estimate of $\mathbf{I}$. The computation of step 2 is split into two terms:

$$\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1); \mathbf{P}) = \hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)) - \hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1) | \mathbf{P}) \,. \tag{8}$$

In our experiments, we noted that a straightforward computation of this expression can result in inconvenient estimation errors. The reason lies in the

different number of traces used to estimate each term on the right side of Eq. (8). $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1))$ is computed with all available traces, say T. The second term is computed as

$$\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{P}) = \sum_{p=0}^{255} \hat{\Pr}(\mathbf{P} = p)\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{P} = p) \tag{9}$$

and for the computation of each summand about $T/256$ traces are used. This difference in the number of traces translates into different estimation accuracies for each term in Eq. (8), burying the small relevant difference between them due to the effect of $\mathbf{P}$ in the larger difference due to the different estimation accuracies.

To amend this, since we have that $\mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1)) = \mathbf{I}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{D})$ for a uniformly distributed *dummy* random variable $\mathbf{D}$ that is independent of the leakages and taking values in $\{0, \ldots, 255\}$, we can compute $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1))$ in a way that resembles Eq. (9) and approximate it by

$$\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{D}) = \sum_{d=0}^{255} \hat{\Pr}(\mathbf{D} = d)\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{D} = d)\,. \tag{10}$$

This leads to equally (in-)accurate estimates for both terms in Eq. (8) and the difference between them is mostly due to the effect of $\mathbf{P}$.

To illustrate the effectiveness of step 2 we compute $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{D})$ and $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1)|\mathbf{P})$ from 50 000 measurements using the same bin distributions for both terms. We use this relatively large number of traces to present aesthetically pleasant figures. Far less traces are sufficient for the method to work.

Figure 1 (left) shows a plot of the values of the first term of Eq. (8), i.e. $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1))$ computed as Eq. (10), for $t_0, t_1 \in \{1, \ldots, 800\}$ and $t_0 \neq t_1$. It is obviously sufficient to compute the values only for $t_0 < t_1$ or $t_0 > t_1$. The x- and y-axes both denote time. We plot a mean trace next to each of them for orientation. The values of mutual information are represented by different colors according to the color bar on the left side. We blank out most pairs of time samples, those that yield small values of mutual information, by plotting them in white. All pairs that yield mutual information values above a certain threshold are plotted in black.

We can see that the locations of the pairs have a clear structure and could possibly aid reverse-engineering of the implementation. Since we know the implementation, we can easily relate parts of the figure to operations: AddRoundKey (approx. index 100 to 150), SubBytes (approx. index 200 to 300), remasking (approx. 300 to 350), four parts of MixColumn (approx. index 350 to 500), AddRoundKey (approx. index 550 to 600), followed by SubBytes and remasking in round two. These pairs are, however, not yet interesting pairs because it is not clear if they can be exploited by an attack (see the discussion of Eqs. (3) and (4)).

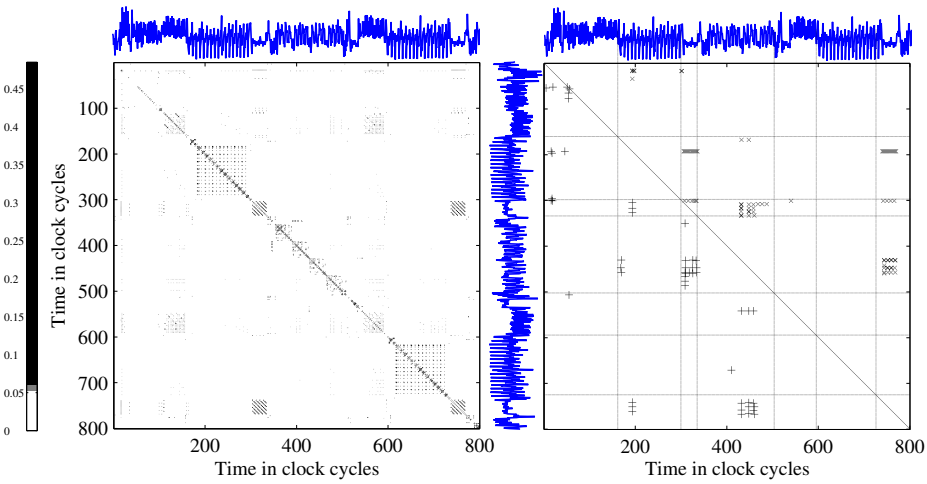

Fig. 1. Left: Matrix of $\hat{\mathbf{I}}(\mathbf{L}(t_0); \mathbf{L}(t_1))$ values. The color bar is in units of bits. A mean trace is plotted next to the axes. Right: Above diagonal, 'x': 100 pairs of time samples where a multivariate MIA attack succeeds. Below diagonal, '+': 100 top ranked pairs in the list of step 2.

Next, we rank the list of pairs according to the result of Eq. (8). The 100 top ranked pairs in the list, i.e. negative mutual information and large absolute value, are depicted in the lower triangle of Figure 1 (right) with '+' symbols.

For the sake of comparison, we include in the upper triangle of the figure the 100 pairs of time samples where a multivariate MIA attack on the third key byte (using the Hamming weight leakage model on predicted S-box output values and 50 000 traces) achieves the largest nearest-rival distinguishing score [24], marked with 'x' symbols.

The partial match between the upper and the lower triangular matrix serves as a first visual evidence for the effectiveness of the method. In particular, the method is able to identify pairs corresponding to both shares of the S-box output of a specific state byte (here the third) without making any hypothesis about the key.

5.4 Performance evaluation of step 2

This section details the performance of the proposed method in finding the pairs that can be exploited for key recovery. Informally, we aim to decouple the performance of the proposed method from the performance of the key recovery attack itself, which is not the focus of this paper. To do so, we first define a set of *good* pairs of time samples that can be attacked and then we analyze the performance of the method in identifying *good* pairs among all possible pairs.

More precisely, we define sets of good pairs by running an attack on all pairs using 50 000 measurements and retaining the 100 resp. 290 pairs that lead

to key recovery and have highest nearest-rival distinguishing score. Our choice for the size of the sets is somewhat arbitrary. The idea is simply to define one smaller set of very good pairs and a larger set that contains additional good pairs with lower nearest-rival distinguishing score. Since different attacks may favor different pairs, we define such sets for three cases: multivariate MIA on the S-box output, Correlation Power Analysis (CPA) [3] with centered product combination function [18] on the S-box output and CPA with same combination function on the S-box input (all using the Hamming weight leakage model). In total, we hence define six sets of good pairs.

Once the sets of good pairs are defined, we run step 2 parametrized by the number of traces. For each number of traces, we repeat the run of step 2 on 100 randomly chosen sets of traces and, each time, keep the position of the best ranked good pair in the list generated by step 2. In other words, we test the pairs in descending order of their ranking (rank 1 is best) and stop as soon as a pair is good. This ranking position is the minimum size of the list from step 2 required for step 3 to succeed in that particular run for a given attack technique. Recall that, by definition, an attack on a good pair succeeds with a comfortable nearest-rival distinguishing score (albeit the absolute margin for a CPA attack on the S-box input is a lot smaller). We hence evaluate only the performance of step 2.

The distributions of the ranks of the best ranked good pairs are shown as boxplots in Fig. 2 (sets of 100 good pairs) and in Fig. 3 (sets of 290 good pairs). For both figures, the used attack techniques are, from left to right: MIA S-box output, CPA S-box output and CPA S-box input.

In the boxplots, the central mark is the median (2nd quartile) and the box edges (solid) represent the 1st and the 3rd quartile. The whiskers (dashed) extend to $q_3 + 1.5(q_3 - q_1)$ and $q_1 - 1.5(q_3 - q_1)$, where q_1 and q_3 are the 1st and 3rd quartiles, respectively. Outliers are marked with '+' symbols.

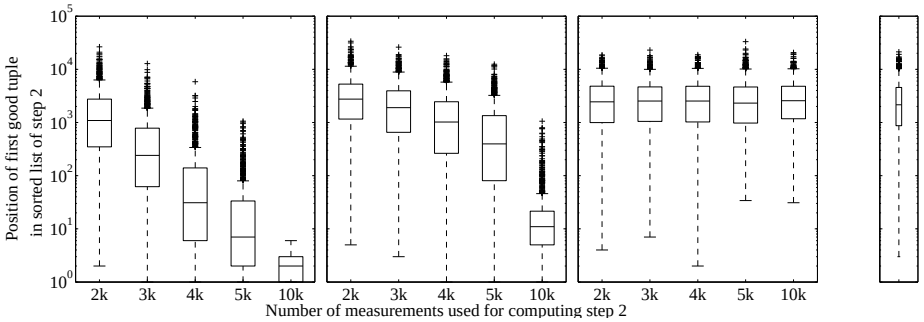

Fig. 2. Distribution of the ranking of the first *good* pair in the list of step 2. Left to right: MIA S-box output, CPA S-box output, CPA S-box input, hypothetical random method. 100 good pairs.

For comparison, the rightmost boxplot in each figure shows the distribution that a hypothetical method that ranks the pairs at random, instead of step 2, would produce. These distributions are independent of an attack technique and only relate to the number of good pairs among all pairs, here 100 resp. 290 out of $800 \times 799/2 = 319\,600$.

One can observe that the proposed method begins to identify good pairs (i.e. to perform better than a random guess) that are exploitable by multivariate MIA or CPA attacks on the S-box output when 3 000 traces or more are available. As the number of traces increases, the medians of the distributions become smaller, i.e. good pairs move steadily toward the top of the list.

One can also observe that our method ranks good pairs for multivariate MIA slightly higher than good pairs for CPA on the S-box output. On the other hand, the method is not able to identify good pairs for a CPA attack on the S-box input better than a random guess. We note that both behaviors are not a property of our method but probably related to our test platform and the implementation.

In the case of larger lists of 290 good pairs, the previously made observations mostly hold. As expected, the medians of the distributions are smaller than in the case of 100 good pairs, simply because even a random guess becomes more likely to succeed. In addition, we can observe that the method now ranks good pairs for multivariate MIA and CPA on the S-box output almost equally well.

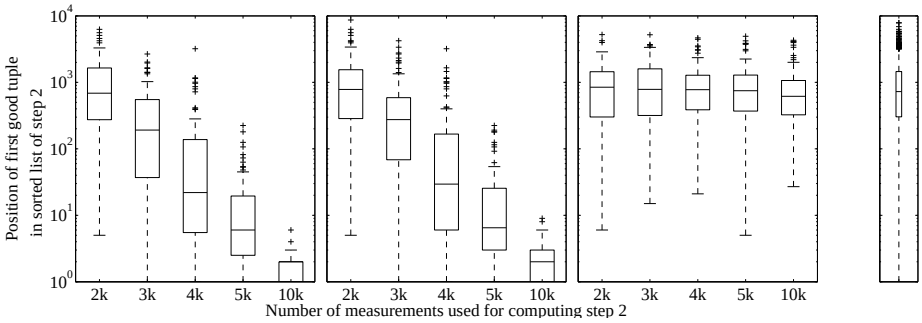

Fig. 3. Distribution of the ranking of the first *good* pair in the list of step 2. Left to right: MIA S-box output, CPA S-box output, CPA S-box input, hypothetical random method. 290 good pairs.

5.5 Practical attacks

The above results highlight important properties of our method and demonstrate that it is effective. In practice, one is however less interested in the exact rank of the first good pair in the sorted list, and more interested in the success rate of an attack end-to-end. This clearly involves the performance of our method *and* the efficiency of the attack used in step 3.

Table 2 shows success rates for steps 2 and 3 together. First we use a given number of randomly chosen traces to compute step 2. Then we attack the $\gamma = 10$ resp. 100 best ranked pairs with multivariate MIA, CPA on the S-box output and CPA on the S-box input (as described before) in step 3, using the same traces. We repeat this procedure 100 times. For the numbers in the first row of the table, we considered an attack successful if the correct key leads to the smallest correlation (or mutual information) value (negative sign and highest absolute value), over all evaluated γ pairs. For the numbers in the second row of the table, we additionally required the correct key to stand out at least by a factor of 1.5 compared to the nearest rival (left) and by a factor of at least 2 (right).

Table 2. Success rates for steps 2 and 3 together, for several parameters: number of traces, size γ of the list of step 2, key recovery attack.

Number of traces		2k	3k	4k	5k	10k		2k	3k	4k	5k	10k
MIA S-box output	$\gamma = 100$	3	15	59	83	100	$\gamma = 10$	0	3	34	53	100
CPA S-box output		11	41	75	95	100		1	11	48	66	100
CPA S-box input		1	0	0	1	0		2	0	0	1	0
MIA S-box output	$\gamma = 10$	0	2	15	35	100	$\gamma = 10$	0	0	7	17	90
CPA S-box output	factor 1.5	0	3	28	52	98	factor 2	0	0	10	17	78
CPA S-box input		0	0	0	0	0		0	0	0	0	0

A first observation is that a CPA attack on the S-box input does not work in our concrete scenarios. CPA attacks on the S-box output converge slightly faster toward 100% success rate than multivariate MIA attacks on the S-box output. We can further see that, given enough traces, both attacks in step 3 eventually reach 100% success, even if we attack only the top ten pairs of step 2 and require the correct key to stand out by a factor of at least 1.5. These results confirm that the combination of steps 2 and 3 works in practice, and that step 2 is able to identify exploitable pairs of time samples. Interestingly, one can further see that multivariate MIA attacks on the S-box output have a small advantage over CPA attacks on the S-box output, if we require the correct key to stand out by a factor of at least 2.

5.6 Computational efficiency

In Tab. 3 we present empirical execution times for our implementations of the proposed workflow (steps 2 and 3) and the strategy that uses exhaustive search instead of step 2. Step 3 of the proposed workflow was performed with multivariate MIA on the S-box output (using the Hamming weight leakage model and list size $\gamma = 100$). For the exhaustive search strategy we evaluated two variants: multivariate MIA on the S-box output (using the Hamming weight leakage model) and CPA on the S-box output (with centered product preprocessing).

All implementations were executed on the same processor on a single core. We note that the absolute execution times are heavily implementation-dependent and thus relative speed-ups are more interesting, since they are less tied to the particular implementation used.

Table 3. Empirical execution times for steps 2 and 3 ($\gamma = 100$) of the proposed workflow and several attacks using exhaustive search.

Number of traces	step 2 + step 3	Exhaustive search	Improvement factor
5 000	2m30s + 2s	MIA-HW 1h48m	43
		CPA 2h 48m	68
50 000	10m24s + 19s	MIA-HW 17h18m	97
		CPA 23h32m	132

A first observation regarding Tab. 3 is the speed-up achieved by the proposed workflow, compared to exhaustive search, when multivariate MIA is used for key recovery. This is a directly interpretable result that corresponds to scenario (b) in Sect. 4.1. The improvement factor in this case is of 43 when 5 000 and 97 when 50 000 traces are used, respectively. We observe that, for our implementations, the factor β/α depends on the number of traces.

One can further see that, for our implementations, applying the proposed workflow is even advantageous if exhaustive search is done with CPA. It achieves an improvement factor of 68 in the running time of the attack when 5 000 traces are used, and an improvement factor of 132 when 50 000 traces are used. However, we stress that this result is not universally valid. The speed-ups are heavily affected by the relative efficiency of our implementations of linear correlation and mutual information estimation.

As a final observation concerning Tab. 3, we remark the validity of the approximation we made in Tab. 1: the running time of the proposed workflow is dominated by step 2. Step 3 contributes at most 3% to the total running time if the list size is $\gamma = 100$.

6 Conclusion

Multivariate DPA attacks can suffer from the effect known as "combinatorial explosion" and hence combinatorial time complexity in the number of time samples that have to be exploited jointly. We presented a novel technique to identify such interesting tuples of time samples before key recovery. Compared to previous work on this topic, our method is not heuristic but systematic and works in black-box conditions using only known inputs (or outputs). Our technique can lead to a substantial improvement in the computational efficiency of multivariate attacks compared to exhaustive search over the same window of time samples,

since it retains only a small fraction of all possible tuples for key recovery. Our approach is based on mutual information and is fully generic, i.e. it does not require more restrictive assumptions than a generic MIA attack. Experimental results based on power traces of a masked software implementation of the AES confirm the effectiveness of the technique, highlight some of its interesting properties and attest attractive running time improvements. An aspect that is not fully explored in this paper and left for future work is a thorough analysis of the number of traces needed for the technique to work.

Acknowledgments

We thank the anonymous reviewers for their thorough evaluation and insightful comments.

This work was supported in part by the Research Council of KU Leuven: GOA TENSE (GOA/11/007), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II, by the Flemish Government FWO G.0550.12N and by the Hercules Foundation AKUL/11/19. Benedikt Gierlichs is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

References

1. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s). In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer, 2002.
2. L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2011.
3. 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. Springer, 2004.
4. 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. Springer, 1999.
5. B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede. Revisiting higher-order DPA attacks: Multivariate mutual information analysis. In J. Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 221–234. Springer, 2010.
6. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008.
7. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Ç. K. Koç and C. Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.

8. C. Herbst, E. Oswald, and S. Mangard. An AES smart card implementation resistant to power analysis attacks. In J. Zhou, M. Yung, and F. Bao, editors, *ACNS 06*, volume 3989 of *LNCS*, pages 239–252. Springer, 2006.
9. M. Joye and F. Olivier. Side-channel analysis. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 1198–1204. 2011.
10. 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. Springer, 2005.
11. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
12. 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. Springer, 1999.
13. T.-H. Le and M. Berthier. Mutual information analysis under the view of higher-order statistics. In I. Echizen, N. Kunihiro, and R. Sasaki, editors, *IWSEC 10*, volume 6434 of *LNCS*, pages 285–300. Springer, 2010.
14. K. Lemke-Rust and C. Paar. Gaussian mixture models for higher-order side channel analysis. In P. Paillier and I. Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 14–27. Springer, 2007.
15. 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. Springer, 2000.
16. E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In D. Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 192–207. Springer, 2006.
17. E. Prouff and M. Rivain. Theoretical and practical aspects of mutual information based side channel analysis. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 499–518. Springer, 2009.
18. E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
19. 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. Springer, 2006.
20. 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. Springer, 2010.
21. A. Venelli. Efficient entropy estimation for mutual information analysis using B-splines. In P. Samarati, M. Tunstall, J. Posegga, K. Markantonakis, and D. Sauveron, editors, *WISTP*, volume 6033 of *LNCS*, pages 17–30. Springer, 2010.
22. N. Veyrat-Charvillon and F.-X. Standaert. Mutual information analysis: How, when and why? In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 429–443. Springer, 2009.
23. 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. Springer, 2004.
24. C. Whitnall and E. Oswald. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 316–334. Springer, 2011.