Efficient Selection of Time Samples for Higher-Order DPA with Projection Pursuits

François Durvaux[1], François-Xavier Standaert[1], Nicolas Veyrat-Charvillon[2]
Jean-Baptiste Mairy[3], Yves Deville[3].

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] IRISA-CAIRN, Campus ENSSAT, 22305 Lannion, France.
[2] ICTEAM/INGI, Université catholique de Louvain, Belgium.

Abstract. The selection of points-of-interest in leakage traces is a frequently neglected problem in the side-channel literature. However, it can become the bottleneck of practical adversaries/evaluators as the size of the measurement traces increases, especially in the challenging context of masked implementations, where only a combination of multiple shares reveals information in higher-order statistical moments. In this paper, we describe new (black box) tools for efficiently dealing with this problem. The proposed techniques exploit projection pursuits and specialized local search algorithms, work with minimum memory requirements and practical time complexity. We validate them with two case-studies of unprotected and first-order masked implementations in an 8-bit device, the latter one being hard to analyze with previously known methods.

1 Introduction

The selection of Points-Of-Interest (POIs) in leakage traces is an important (and not very discussed) problem in the application of Side-Channel Analysis (SCA) attacks. When targeting unprotected implementations, the naive strategy that is commonly used in the literature is to test all the time samples independently. It raises two important challenges. First, how to combine these time samples efficiently, in order to maximize the amount of information extracted from each leakage trace? Second, how to extend this technique in the context of masked implementations where the sensitive data is split into d shares manipulated in different clock cycles (as it is typically the case in software), and only the combination of these shares' leakage reveals key-dependent information – which makes the complexity of an exhaustive analysis grow combinatorially with d?

Solutions to the first problem typically include dimensionality reduction techniques such as PCA and LDA. These tools (introduced to SCA in [1, 23] and recently revisited in [3, 5]) essentially project the leakage traces into a lower-dimensional subspace that optimizes some *objective function*. Namely, PCA usually maximizes the variance between the mean leakage traces – i.e. the signal of a first-order DPA, while LDA maximizes the ratio between inter-class and intra-class variances – i.e. its Signal-to-Noise Ratio (SNR), essentially. Their main advantage is to provide a principled and intuitive solution to the problem, since the projection (i.e. eigenvectors) they produce indicate the POIs. Yet, they are somewhat limited when moving to masked implementations for which the information lies in high-order statistical moments, since their objective function is based on a definition of signal that primarily captures first-order leakages[1].

Solutions to the second problem are even sparser. To the best of our knowledge, the usual reference for selecting POIs for masked implementations is the educated guess proposed by Oswald et al. in [13] (i.e. an exhaustive search over all d-tuples of time samples in a window selected based on engineering intuition). Next, Reparaz et al. proposed an alternative solution exploiting Mutual Information Analysis (MIA) [8], that allows gaining a constant (but practically meaningful) factor corresponding the number of key hypotheses in the attack [20]. In both cases, the proposed tools do not output a projection but a list of the most useful POIs (i.e. d-tuples) in function of the (non-profiled) attack considered.

In this paper, we investigate the use of *Projection Pursuits* (PPs), as alternative tools for the selection of POIs in leakage traces [7]. Intuitively, PPs machine-pick "interesting" low-dimensional projections of a high-dimensional data space by numerically maximizing a certain objective function. They essentially work by tracking the improvements (or lack thereof) of the projection when modifying it with small random perturbations. Their main advantage in our context is that they can deal with any objective function, which naturally fits to the problem of higher-order SCA. Their main drawback is (in general) their heuristic nature, since the convergence of the method is not guaranteed and its complexity is context-dependent. As a result, and in order to validate the interest of PPs in our SCA context, we first applied them to the simple case of an unprotected implementation of the AES. We show that different objective functions can be efficiently used for this purpose, leading to powerful subspace-based attacks, with similar informativeness as previous solutions such as LDA.

Next, we moved to the more challenging context of masking. In this case, we combined the (linear) projection with an objective function exploiting higher-order statistical moments. Initial experiments suggest that the straightforward implementation of a PP algorithm is not efficient in detecting the POIs of such protected implementations (especially as the number of useless dimensions in the traces increases). The main reason is that as long as a d-tuple of POIs is not present in the projection, the objective function essentially returns random indications. Interestingly, we then show that a specialized PP algorithm exploiting an improved *local search* could give excellent results even in this challenging context. Intuitively, it works by looking for the best size and position of d windows covering parts of the traces, again by iterating small random perturbations. Our experiments suggest that we can recover POIs with significantly less calls to the objective function than a exhaustive analysis. We further discuss the main parameters influencing the success of such a detection method, and detail the time vs. measurement complexity tradeoff resulting from these parameters.

[1] Of course, a trivial solution would be to apply PCA/LDA to "product traces" containing all the possible products of d-tuples, but this rapidly leads to unrealistic memory requirements in the masked software context that we consider next.

2 Background

Notations. We use capital letters for random variables, small caps for their realizations, sans serif fonts for functions and calligraphic letters for sets.

2.1 Measurement setups

Our experiments are based on measurements of an AES implementation run by an 8-bit Atmel AVR (ATMega644P) microcontroller at a 20 MHz clock frequency. We monitored the voltage variations across a 22 Ω resistor introduced in the supply circuit of our target chip. Acquisitions were performed using a Lecroy HRO66ZI oscilloscope running at 200 MHz and providing 8-bit samples. For concreteness, our evaluations focused on the leakage of the first AES master key byte (but would apply identically to any other enumerable target). Leakage traces were produced according to the following procedure. Let x and s be our target input plaintext byte and subkey, and $y = x \oplus s$ denote a key addition. For each of the 256 values of y, we generated 1000 unprotected encryption traces (resp. 500 for masked traces), where the rest of the plaintext and key was random, i.e. we generated 256 000 (resp. 128 000) traces in total, with plaintexts of the shape $p = x||r_1||\ldots||r_{15}$, keys of the shape $k = s||r_{16}||\ldots||r_{30}$, and the r_i's denoting uniformly random bytes. In case of masked implementations, additional uniform randomness was used to generate the shares. In order to reduce the memory cost of our evaluations, we only stored the leakages corresponding to the 2 first AES rounds in the unprotected case (as the dependencies in our target byte $y = x \oplus s$ typically vanish after the first round, because of the strong diffusion properties of the AES). As for the protected case, we only considered a single S-box, for which the precomputation of a masked table alreay implies large traces with $Ns = 30,000$ time samples (vs. $Ns = 1500$ for the unprotected one). As will be clear next, these sets of measurements were large enough to emphasize the interest of our projection pursuit algorithms. In the following, we will denote the 1000 (resp. 500) encryption traces obtained from a plaintext p including the target byte x under a key k including the subkey s as: $\mathsf{AES}_{k_s}(p_x) \rightsquigarrow l_y^i$, with $i \in [1;1000]$ (resp. $i \in [1;500]$). Whenever accessing the points of these traces, we will additionally use an argument t (for time), leading to $l_y^i(t)$. Our goal is to generate projections exhibiting the time samples that contain information about y. Note that since we assume the plaintext to be known by the adversary (as usual in SCAs), it directly translates into information about s – which typically occurs during the key addition $y = x \oplus s$ and S-box execution $z = \mathsf{S}(x \oplus s)$.

2.2 Objective functions (aka evaluation metrics)

In order to "guide" the PP, we need to define criteria to determine whether some modification of the projection is positive. Any SCA evaluation metric can be used for this purpose. We list a few candidates in this section. In order to guarantee their soundness, we focused on objective functions based on profiled distinguishers (which allows mitigating biases due to incorrect a-priori choices of models – given that the profiles are well estimated and based on sound assumptions).

CPA [4]. In a profiled Correlation Power Analysis, the adversary first estimates the first-order moments corresponding to each value y from a vector of N_p profiling traces $\mathbf{l}_y^{\mathsf{p}}$, that we denote as $\hat{\mathbf{m}}_y^1 = \hat{\mathsf{E}}(\mathbf{l}_y^{\mathsf{p}})$, with $\hat{\mathsf{E}}$ the sample mean operator. This step is performed for each time sample independently, leading to $\hat{\mathbf{m}}_y^1(t)$. Since there are 256 y values in our AES case study, it amounts to compute $256 \times N_s$ means, with N_s the number of samples per trace. Then, he computes the correlation between these mean values and the samples coming from a vector of test traces $\mathbf{l}_y^{\mathsf{t}}$, leading to $\hat{\rho}(\hat{\mathbf{m}}_y^1(t), \mathbf{l}_y^{\mathsf{t}}(t))$ with $\hat{\rho}$ denoting Pearson's coefficient.

SNR [10]. An alternative to CPA is the SNR defined at CT-RSA 2004 as:

$$\hat{\mathsf{SNR}}(t) = \frac{\hat{\mathsf{var}}_y\Big(\hat{\mathsf{E}}\big(\mathbf{l}_y^{\mathsf{t}}(t)\big)\Big)}{\hat{\mathsf{E}}_y\Big(\hat{\mathsf{var}}\big(\mathbf{l}_y^{\mathsf{t}}(t)\big)\Big)},$$

with $\hat{\mathsf{var}}$ the sample variance operator. Similarly to the correlation coefficient, such a criteria is discriminant for first-order information (i.e. information lying in the first-order moments of the leakage distribution). In order to deal with masked implementations, we also need objective functions that capture more general dependencies. In this context, a natural option is the information theoretic metric introduced in [25] and later refined in [19]. Its sample definition is given by:

$$\hat{\mathrm{I}}(S;X,L) = \mathrm{H}[S] - \sum_{s\in\mathcal{S}} \Pr[s] \sum_{x\in\mathcal{X}} \Pr[x] \sum_{l_y^i\in\mathcal{L}_Y^t} \Pr\nolimits_{\mathtt{chip}}[l_y^i|s,x].\log_2 \hat{\Pr}\nolimits_{\mathtt{model}}[s|x,l_y^i],$$

where $\hat{\Pr}_{\mathtt{model}}$ is a probabilistic model estimated thanks to the set of profiling traces (just as the $256 \times N_s$ mean values in the correlation case). Computing such an objective function implies (constant but significant) performance overheads, because it requires applying Bayes' law and marginalizing over the key hypotheses. Since the objective function will typically be applied after projection in the following sections (i.e. in a univariate context), a cheaper alternative is to exploit the following "Moments-Correlating Profiled DPA" (MCP-DPA):

MCP-DPA [12]. The attack features essentially the same steps as a profiled CPA. The only difference is that the adversary will estimate dth-order moments $\hat{\mathbf{m}}_y^d(t)$ with the profiling traces. In the following, we will be particularly interested in the Moments against Moments Profiled Correlation (MMPC) criteria:

$$\mathrm{MMPC}(t) = \hat{\rho}(\hat{\mathbf{m}}_y^d(t), \tilde{\mathbf{m}}_y^d(t)),$$

where $\tilde{\mathbf{m}}_y^d(t)$ are another vector of moments, estimated with the test traces. As detailed in [12], MCP-DPA is able to capture information in any statistical moment, while enjoying the implementation efficiency of CPA (which is highly beneficial in our context where the objective function is intensively used).

3 Projection pursuit against unprotected devices

In this section we investigate the application of PPs to the simple case of the (unprotected) AES furious implementation available as open source from [15]. In this context, our goal is to find a projection vector $\boldsymbol{\alpha}$ that will convert the N_s samples of a leakage vector $\mathbf{l}_y^{\mathsf{t}}$ to a single (projected) sample λ_y^i, that is:

$$\lambda_y^i = \sum_{t=0}^{N_s-1} \alpha(t) \cdot l_y^i(t),$$

such that univariate attacks exploiting the λ_y^i's will be most efficient. This essentially requires to define an objective function that measures the "informativeness" of these samples. As mentioned in the previous section, this task is quite easy when first-order information is available in the leakage traces: Pearson's correlation coefficient obtained from a CPA and Mangard's SNR are natural candidates – we will try them both in the next subsection. Following the equivalence results in [11], they should provide similar results in this case (also similar to the ones that would be obtained with an information theoretic metric).

3.1 Projection pursuit algorithm

The pseudo-code of our projection pursuit algorithm is given in Algorithm 1.

Algorithm 1 Projection Pursuit.

```
basic_PP($N_r$,$N_{it}$)
$\boldsymbol{\alpha} = initialize()$;
repeat $N_r$ times
    $r = rand\_index(N_s)$;
    $\boldsymbol{\alpha}_{new} = max\_search(@f_{obj}, \mathcal{L}^{\mathsf{p}}, \boldsymbol{\alpha}, r, N_{it})$;
    $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{new}$;
end
```

It essentially repeats (N_r times) the selection of a random index r followed by a maximization of the objective function for the corresponding time sample, based on the set of profiling traces $\mathcal{L}^{\mathsf{p}}$ (which contains traces for all the intermediate values y). For this purpose, the *max_search*() function consists in successive parabolic interpolations (illustrated in the long version of this work [6]), which work in two iterated steps. We first look for samples that enclose the extremum as follows. From a starting point x_1, we add a Δ in the direction that increases f_{obj} to get x_2. Then, we keep adding Δ's until finding x_3 such that $y_3 < y_2$. As the weights assigned to each time sample are between 0 and 1, we typically take Δ's corresponding to a couple of percents (e.g. 0.1 in our experiments) and repeat such additions at most $1/\Delta$ times. Then, based on these three points, we start interpolating. This process is iterated N_{it} times, during which we replace the "oldest" x-point by the x-coordinate (x_v) of the parabola vertex (y-values are re-computed accordingly). The new $\alpha(t)$ gets its value from the median x-value at the end of the last iteration. In our experiments, $N_{it} = 3$ iterations were enough to get a good approximation of the maximum. This method has the advantage of being very fast to compute and to converge. Note finally that the number of repetitions N_r should ideally be larger than the number of samples N_s (e.g. twice, typically), because some weights benefit from being re-adjusted after the modification of other $\alpha(t)$'s. Yet, when applied in the context of an unprotected implementation, the time complexity of Algorithm 1 was never a practical limitation in our experiments (i.e. a few minutes of computations).

3.2 Experimental results

We implemented the PP algorithm for both the CPA and SNR objective functions, and targeted the first AES key byte for illustration. For each of the 256 values of $y = x \oplus s$, we measured $N_p = N_t = 50$ traces for the CPA objective function, and $N_t = 100$ traces for the SNR one, each of them made of $N_s = 1500$ time samples. We set N_r, N_{it} and Δ as just explained (to 3000, 3 and 0.1, respectively). We then computed success rates to compare the quality of the projections obtained with the most informative sample, by performing 2000 experimental univariate Template Attacks (TA). These results show the effectiveness of the projections as they need only 7 traces to get a 90% success rate, against 28 traces for the univariate TA. It also confirms that both objective functions are indeed equivalent in this case. It is finally interesting to compare our findings with the results in [24] that target a similar implementation (with very similar success rate for the univariate TA). In particular, we see that the univariate attack based on the single sample provided by our projections leads to approximately the same data complexities as the hexavariate template attack taking (heuristic) advantage of all the POIs in this previous work.

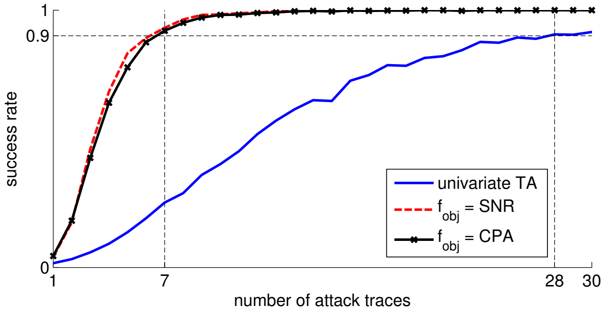

Fig. 1. Template attack success rates against unprotected device

4 Projection pursuit against masked implementations

In contrast with the previous section, detecting POIs in leakage traces of masked implementations is a quite challenging task. From the complexity point-of-view, exhaustive approaches may grow exponentially with the number of shares (if these shares are manipulated at different time samples), making them unpractical for long traces. Furthermore, the information in the leakages of masked implementation lies in higher-order moments of their probability distribution, which are harder to estimate. As a result, the direct application of Algorithm 1 with the previous objective functions in this context does not provide successful results. In the (simple) case where the shares of a masking scheme are manipulated in parallel, adapting the objective function may be sufficient to deal with this problem. But in case of software implementations, where the shares are manipulated at different time samples, it is the algorithm itself that has to be adapted. Intuitively, this is because it works by modifying time samples one at a time, while for such masked implementation, we require at least one meaningful d-tuple of samples to be active in the projection for an objective function to output relevant information. We now describe how to specialize PPs to take this constraint into account, and detect POIs for masked implementations.

4.1 Specialized projection pursuit algorithm

The main tool used in our following optimization is local search, which is a collection of iterative methods that are efficient for quickly finding good solutions to optimization problems (note that the previous PP algorithm can be viewed as a simple local search). Despite heuristic, it generally works more efficiently than exhaustive analyses. Furthermore, local search has very limited storage requirements. For example, in our context, it exploits the leakage traces directly – which is a significant advantage compared to heuristics exploiting "product traces" as mentioned in footnote 1. A good reference to these methods is [9]. Their working principle is simple: they always keep a solution (called the current solution) as well as the best solution found since the beginning of the search. At each iteration of the algorithm, the current solution is perturbed, giving a set of new solutions, called its neighborhood. One of the neighboring solutions is then selected and replaces the current solution. The algorithm terminates when its convergence criterion is met (e.g. number of iterations without improvement, time limit, etc.). Intuitively, such an approach to optimization exploits *diversification* and *intensification*. The former aims at exploring a large and diverse search space, while the latter intends to improve the current solution. Their combination is expected to avoid being trapped into local optima

When applied to masking, one key element has to be taken into account by optimizations. Namely, the sensitive variables are split into d shares and the objective function should not be informative as long as a meaningful d-tuple of shares is not present in the projection. Besides, in practice it frequently happens that dimensions near a POI also contain valuable information. These two facts motivate the way we designed our improved search algorithm as follows. First, we consider a projection vector containing d *windows* of non-zero weights (all the others being zero) and denote a group of successive dimensions as a window. The weights inside these windows are uniform. In this context, and since local search only considers local modifications of the current solution, the information given by the objective function will return essentially random indications (so no reliable information) if this current solution does not cover the d shares. On the contrary, when the windows spans a d-tuple of shares, the objective function can be used to refine the current solution. For this reason, our specialized PP algorithm will be split into two parts next denoted as `find_sol` and `improve_sol`. The `find_sol` phase probes the search space with large windows and a lot of randomness until it has good indication that the windows span the d-tuples of shares. In order to detect that the windows span these d-tuples, we use two sets of profiling traces ($\mathcal{L}^{\mathsf{p}}_{\mathsf{tr}}$ and $\mathcal{L}^{\mathsf{p}}_{\mathsf{va}}$, where tr stands for training and va for validation). Then, the `improve_sol` phase refines those windows. The `find_sol` phase thus puts more emphasis on diversification and the `improve_sol`, on intensification.

Algorithm 2 Specialized projection pursuit algorithm using local search.

```
specialized_PP_Local_Search($d, W_{len}, T_{det}$,TP:=TP'∪ TP'')
    $\boldsymbol{\alpha}$ = find_sol_phase($d, W_{len}, T_{det}$,TP');
    if($\boldsymbol{\alpha} \neq null$)
        return improve_sol_phase($\boldsymbol{\alpha}$,TP'');
    end
```

The pseudocodes of the specialized PP algorithm using local search are given in Algorithms 2, 3 and 4. These algorithms depend on various parameters: some of them will be explicitly discussed as they hold important intuitions, the remaining ones – next denoted as technical parameters (TP) – will be fixed according to state-of-the-art strategies. Our main tool is the `specialized_PP_Local_Search` function (Algorithm 2). As just explained, it organizes the search in two main steps. The first one is the `find_sol` phase which returns a first candidate projection $\boldsymbol{\alpha}$ (after N_r^f repetitions). If this first step is successful, the `improve_sol` phase is repeated N_r^i times to refine the solution. The `find_sol` phase is described in Algorithm 3. At each iteration, it randomly selects d windows of length W_{len} with non-zero weights (function `random_window`). All the neighbors of the solution are then computed with the function `get_neighbors_FS`. Each neighbor is constructed by moving one of the windows left or right (if we see the projection vector as a row vector). The lengths of the moves considered are small multiples of the window length (as set by the *num_hops* parameter). During the computation of the neighbors, the collisions between windows are avoided in order to keep d distinct windows. Next, the best neighbor is selected as the neighbor having the maximal evaluation of f_{obj} on the set $\mathcal{L}^{\mathsf{p}}_{\mathsf{tr}}$. This best neighbor is finally tested to detect if a d-tuple of shares is spanned by the windows. The detection is based on a threshold T_{det} on the objective function that will be

Algorithm 3 Find solution phase.

```
find_sol_phase($d, W_{len}, T_{det}$,TP')
TP':={$N_r^f$, $num\_hops$}
    i=0;
    repeat $N_r^f$ times
        $\boldsymbol{\alpha} = random\_window(d, W_{len})$;
        $neighborhood = get\_neighbors\_FS(\boldsymbol{\alpha}, num\_hops)$;
        $best\_neighbor = max(@f_{obj}, neighborhood, \mathcal{L}^{\mathsf{p}}_{\mathsf{tr}})$;
        if $f_{obj}(best\_neighbor, \mathcal{L}^{\mathsf{p}}_{\mathsf{tr}}) > T_{det}$ & $f_{obj}(best\_neighbor, \mathcal{L}^{\mathsf{p}}_{\mathsf{va}}) > T_{det}$
            return $(i+1, best\_neighbor)$;
        end
        i++;
    end
end
```

carefully discussed in the next section. In order to dodge the randomness of the objective function when the d shares are not spanned, this threshold has to be exceeded on both the training and validation sets of traces $\mathcal{L}^{\mathsf{p}}_{\mathsf{tr}}$, $\mathcal{L}^{\mathsf{p}}_{\mathsf{va}}$. If those two conditions are met, the projection vector is returned by the algorithm.

If the `find_sol` phase was able to find a solution spanning the d shares, the objective function is informative enough to allow a second (intensification) step, and the `improve_sol` phase (in Algorithm 4) is run for N_r^i iterations. At each iteration, the entire neighborhood is constructed with the function `get_neighbors_IS`. Each neighbor results from the shift (left or right) of one window or the resizing of all the windows (we keep the same size for all windows). The move steps considered are given in *move_steps*, and the resize steps in *resize_steps*. The size of the windows is constrained to remain between *min_WS* and *max_WS*. The selection of the neighbor is then performed by `select_neighbor`, as a random neighbor amongst the N_n best neighbors. Using this selection strategy allows the search to avoid being trapped into local optima, ensuring a sufficient diversification. The search also memorizes the best projection obtained since the beginning of the phase in $\boldsymbol{\alpha}_{best}$. This is mandatory as it is allowed to select projection vectors that decrease the objective function. Eventually, the variable *num_stagn* records the number of iterations without improvement of the best solution. Once *num_stagn* is larger than *max_stagn* or when the number of iterations reaches $N_r = N_r^f + N_r^i$, the search returns the best solution $\boldsymbol{\alpha}_{best}$.

As far as the technical parameters are concerned, we first set the number of hops (*num_hops*) in the `find_sol` phase to allow the windows covering all the dimensions of the traces. It enables an iteration to find a covering set of windows when one window is incorrectly placed. Next, in the `improve_sol` phase, the more move steps (*move_steps*) and resize steps (*resize_steps*), the quicker the algorithm converges towards the optimal windows, but the longer each iteration is. We found that a good tradeoff in our context was to use *move_steps* of 1, 3 or 5 dimensions and *resize_steps* of 1 dimension. Those settings allow the

Algorithm 4 Improve solution phase.

```
improve_sol_phase($\boldsymbol{\alpha}$,TP'')
TP'':=$\{N_r^i, move\_steps, resize\_steps, minWS, maxWS, N_n, max\_stagn\}$
    $\boldsymbol{\alpha}_{best} = \boldsymbol{\alpha}$;
    Repeat $N_r^i$ times
        $neighborhood = get\_neighbors\_IS(\boldsymbol{\alpha}, move\_steps, resize\_steps, minWS, maxWS)$;
        $\boldsymbol{\alpha} = select\_neighbor(@f_{obj}, \mathcal{L}_{\mathsf{tr}}^{\mathsf{p}}, N_n)$;
        if $f_{obj}(\boldsymbol{\alpha}, \mathcal{L}_{\mathsf{tr}}^{\mathsf{p}}) > f_{obj}(\boldsymbol{\alpha}_{best}, \mathcal{L}_{\mathsf{tr}}^{\mathsf{p}})$
            $\boldsymbol{\alpha}_{best} = \boldsymbol{\alpha}$;
            $num\_stagn = 0$;
        else
            $num\_stagn++$;
        end

        if $num\_stagn > max\_stagn$
            return $\boldsymbol{\alpha}_{best}$;
        end
    end
    return $\boldsymbol{\alpha}_{best}$;
end
```

iterations to be fast while still covering a large part of the search space around the solution found by the `find_sol` phase. The *min_WS* parameter typically depends on the sampling rate of the oscilloscope used in the attack: we set it to 5 which corresponds to half a cycle in our experiments, based on the intuition that dimensions next to a POI may also contain information. *max_WS* was then chosen as 2*W_{len}, reflecting that this information can be spread on multiple clock cycles. Finally, a *max_stagn* value of 50 allows the local search to stop when it is unlikely to further improve the quality of the windows. And given the low span of the moves and the resizes, an exploration parameter N_n of 3 is enough to escape local optima and still converge towards the optimal solution.

4.2 Simulated experiments

We now discuss the setting of the more intuitive parameters W_{len} and T_{det} together with the performance gains obtained thanks to our specialized PP algorithm. In view of their heuristic nature, these questions are best investigated with simulated examples, where we can play with some important parameters of leaking implementations. For this purpose, we will consider a first-order masked S-box where the adversary receives N_i pairs of leakage variables of the form:

$$L_i^1 = \mathsf{HW}(\mathsf{S}(x \oplus s) \oplus m) + R_i^1,$$
$$L_i^2 = \mathsf{HW}(m) + R_i^2, \tag{1}$$

where HW is the Hamming weight function, S the AES S-box, x a plaintext byte, s a key byte, m a secret random mask, and R_i^1, R_i^2 are normally distributed noise variables with variance σ_n^2 ($1 < i \leq N_i$). For simplicity, we make sure that the N_i samples corresponding to the two shares are not overlapping. Next to these $2 \times N_i$ informative samples, we finally add $N_s - 2 \times N_i$ random samples N_j, so that N_s is the total number of samples in our simulated traces.

Setting the detection threshold An important parameter in Algorithm 3 is the threshold value used to decide whether an improvement of the objective function is significant. In this context, a particularly convenient feature of the MMPC criteria (defined in Section 2.2) is that it gradually tends to one as the number of measurements used in the detection increases. That is, given that the order of the statistical moment (e.g. $d = 2$ in our current simulations) and number of measurements used in the detection is sufficient, this criteria always reaches high values. Intuitively, it is because the MMPC relates to the statistical confidence we have in our estimated moments rather than their informativeness (see [12] for a discussion). As a result, and using such an objective function, we are able to set the detection threshold T_{det} in a completely black box manner (i.e. independent of the implementation details). Indeed, the only thing we have to guarantee is that the MMPC as computed by the objective function is significant in front of the one that would be obtained by chance, for non-informative samples. But this essentially depends on the size of the target operations. For example, the correlation between random 256-element vectors is (roughly) Gaussian-distributed[2] with mean zero. And the probability that MMPC > 0.2 by chance in this case is already below the one corresponding to three σ's (i.e. below 0.1%). Of course, one can expect slight deviations from such an ideal behavior (e.g. so-called ghost peaks leading to non-zero mean MMPC for non-informative samples), but our next experiments will confirm that setting T_{det} to 0.2 is generally good.

Impact of W_{len}, σ_n^2 and N_i on the detection success. Given a detection threshold set as just explained, we can now evaluate the impact of different parameters on the success of our *find_sol* phase. In particular, the noise variance σ_n^2, number of informative pairs of samples in the traces N_i and window length W_{len} are important in this respect. As just explained, we know that given a large enough number of measurements, the MMPC criteria should become larger than 0.2 for the informative samples. But it also means that if this number of measurements is not sufficient, the moments used in MCP-DPA will not be sufficiently well estimated and the detection may fail. As usual, the main parameter influencing the estimation complexity is the noise variance σ_n^2. Yet, since we apply the objective function after projection in our PP algorithm, the size of the window W_{len} also matters here. Indeed, adding W_{len} samples with noise variance σ_n^2 implies a larger noise variance $W_{len} \times \sigma_n^2$ after projection. This is typically illustrated in the left part of Figure 2, where we see the impact of increasing W_{len} for two noise levels ($\sigma_n^2 = 0.1$ in the top figure, $\sigma_n^2 = 2$ in the bottom one). That is, for too large noise variances or window lengths, the estimation of the MMPC criteria is not good enough to take good decisions (i.e. is below T_{det}). In other words, more measurements are needed in this case for the PP algorithm

[2] More precise estimates can be obtained with Fisher's Z transform.

to output meaningful results. Interestingly, we also see in the right part of the figure that adding meaningful samples in the traces (i.e. increasing N_i) quite significantly mitigates the impact of large window lengths. So intuitively, traces with multiples POIs available will better benefit from our proposed method.

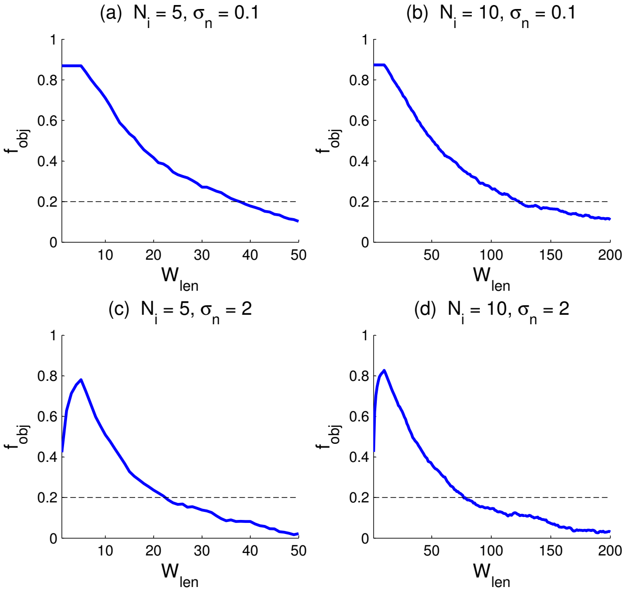

Fig. 2. Incidence of the window length W_{len} on the information detection.

Time complexity. The previous results suggest that the complexity of PP algorithms is essentially a tradeoff between time and measurement complexities. That is, increasing the windows length should decrease their time complexity[3], but increases the noise after projection, and so the number of measurements needed to estimate the MMPC criteria with sufficient confidence. This is typically illustrated in the left part of Table 1, where we also see the benefit of having more informative samples in the traces (i.e. increasing N_i). Furthermore, the right part of the table highlights the impact of increasing the size of the traces N_s. As in a combinatorial search, the time complexity of the PP algorithm should increase quadratically with it (more generally, it depend on N_s^d with d the number of shares in the masking scheme). Yet, increasing W_{len} or N_i can make this increase quasi-linear for some (not too large) values of N_s. Besides, note that Table 1 includes all the constant factors related to the technical parameters in the previous section, which sometimes amortizes these asymptotic predictions. Note

[3] At most linearly since the benefit of increasing the window length W_{len} saturates whenever it is not negligible in front of the number of samples in the traces N_s.

also that this table counts the calls to the objective function for readability, but this count is not fully reflective of the PP's time complexity when changing the size of the profiling sets $\mathcal{L}^{\mathsf{p}}_{\mathsf{tr}}$ and $\mathcal{L}^{\mathsf{p}}_{\mathsf{va}}$, since larger sets also increase the complexity of each evaluation of the objective function. Yet, thanks to the parallelism of MCP-DPA attacks, the impact of these increases was limited in our experiments, leaving us with strong concrete results, as the next section will show.

Table 1. Impact of W_{len}, N_i and N_s on the average number of f_{obj} calls.

$N_s = 1000$		N_i	
		5	10
W_{len}	10	7306	4681
	20	3920	3008
	30	3266	2782
	50	-	2138
	100	-	1020
	150	-	-

	N_s		
	500	1000	2000
$W_{len} = 50, N_i = 10$	905	2138	4673

4.3 Measured experiments

The previous simulated experiments suggest that a specialized PP algorithm can be an efficient way to find POIs in the leakage traces of masked implementations. We now would like to confirm this hope in front of a real case-study. For this purpose, we will consider the actual measurements of a first-order masked AES S-box based on table lookups [17, 22]. For every pair of input/output masks (m, q), it pre-computes an S-box S^* such that $\mathsf{S}^*(x \oplus s \oplus m) = \mathsf{S}(x \oplus s) \oplus q$ Since this pre-computation is part of the adversary's measurements, it leads to quite memory-consuming traces of $N_s = 30,000$ samples (which would be a challenging target for a combinatorial search). Furthermore, we verified empirically that our implementation does not lead to any (easy-to-detect) first-order information leakage, by running template attacks for all the time samples, and making sure that the success rate remained negligible (which should be guaranteed by the use of independent masks m and q, in order to prevent leakages based on the transitions between the the S-box input and output). Our motivation for using this setup was twofold. First, we selected a masking countermeasure based on pre-computed tables in view of the difficulty to obtain a first-order secure implementation based on other standard masking schemes such as [21] – see [2] for a recent discussion of this problem. Second, we purposely put ourselves in a challenging scenario with large traces, without trying to compress them (e.g. by reducing the sampling frequency or through educated guess). While we agree that concrete adversaries would try to exploit these possibilities, we assume that they would not always be able to compress traces up to feasible combinatorial search, and the experiments in this section aim to reflect this possibility.

We then analyzed our set of profiling and test traces, in order to evaluate the success and efficiency of our POI detection tool. We used the same MMPC criteria and detection threshold of 0.2 as previously discussed, and selected a window length W_{len} of 25, corresponding to approximately two clock cycles in our measurements: this is the only physical intuition used in our experiments. With these parameters, it turned out that the estimation of the objective function was sufficiently accurate (for our detection threshold to make sense) with 50 profiling traces per template (i.e. 50×256 among the 500×256 measured). Based on our 1500 test traces, we then evaluated that the local search algorithm was able to return a solution within an average of 12 000 calls to f_{obj} (roughly corresponding to 7 minutes of execution time on our desktop computer). We then repeated this search multiple times in order to find several pairs of informative windows. We finally used these windows to launch multivariate (Gaussian) template attacks using 2, 4 and 8 dimensions. For this purpose, we selected the smallest windows (which turned out to contain 5 samples) and built templates for their mean values (so that each pair of window provided us with 2 dimensions). The results of these attacks are illustrated in Figure 3 and confirm that our tool successfully detected POIs in this challenging case[4]. Interestingly, we see that the gain due to increased dimensionalities vanishes when moving from 4-dimension templates to 8-dimension ones. We conjecture that this mainly relates to template estimation issues. Note anyway that, as mentioned in introduction, these attacks are not aimed to be optimal from the data complexity point-of-view (since we have no guarantee to find the most informative samples). Our main goal was to provide a time-efficient POI detection tool, in a black box setting. To the best of our knowledge, previous methods for this purpose would not have been able to deal with 30,000-sample traces without an educated guess (For illustration, the product traces mentioned in footnote 1 would correspond to 900.10^6 samples).

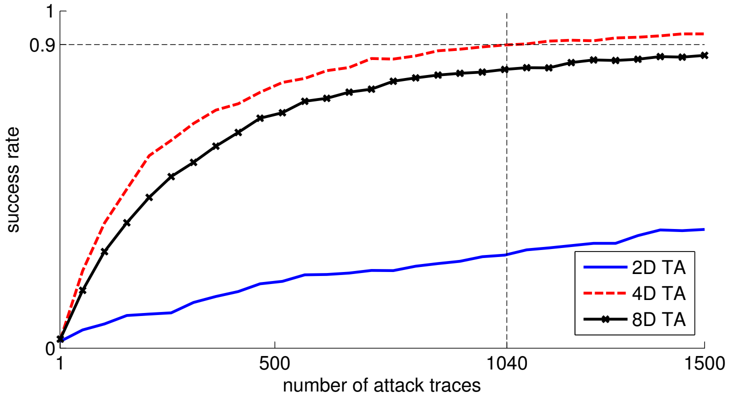

Fig. 3. 4th-order success rates of multivariate template attacks.

[4] For convenience, and in order to limit our measurement needs, we estimated a 4th-order success rate which corresponds to an adversary able to enumerate 2^{32} keys.

5 Conclusions

In this work we proposed an efficient method for finding POIs in the leakage traces of cryptographic implementations. We exploit a combination of PP and local search for this purpose, and discussed the how to adapt it to the side-channel cryptanalysis problem. One of the main advantages of the method is its genericity, as it can be applied to any implementation, by simply adapting its objective function. Besides, it has very low memory requirements compared to state-of-the-art solutions and (although heuristic) works in practical time complexity. We applied our basic and specialized PP algorithms to two case studies of unprotected and 2-share masked implementations to validate our claims. Extending the specialized version to more shares would be straightforward, since this number of shares (i.e. d) is a parameter in our search algorithms.

Among the interesting open problems, we believe investigating the informativeness of the projected samples obtained with PPs in the context of protected implementations is promising – it was essentially left out of our analysis so far. Different approaches could be considered for this purpose. One would be to refine the projection vectors, possibly based on an information theoretic objective function that would better reflect the resulting attacks' data complexity. An alternative one would be to exploit non-linear projections, e.g. inspired by the "product combining" frequently used in second-order DPA [18, 26]. Yet, preliminary results suggest that non-linear projections may be hard(er) to exploit because the addition of non-informative samples when computing the objective function has higher impact on the (non-Gaussian) noise in this case. Besides, testing new objective functions that are cheap to compute and estimate, in the profiled and non-profiled settings, is another interesting research direction.

Note finally that the long version of this work includes two additional subsections with detailed discussions and comparisons with related works [6].

Acknowledgements. F.-X. Standaert is a research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the European Commission through the ERC project 280141 (CRASH).

References

1. Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006.
2. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. *IACR Cryptology ePrint Archive*, 2014:413, 2014.
3. Lejla Batina, Jip Hogenboom, and Jasper G. J. van Woudenberg. Getting more from PCA: First results of using principal component analysis for extensive power analysis. In Orr Dunkelman, editor, *CT-RSA*, volume 7178 of *LNCS*, pages 383–397. Springer, 2012.

4. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
5. Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *LNCS*, pages 253–270. Springer, 2013.
6. François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. *IACR Cryptology ePrint Archive*, 2014:412, 2014.
7. J.H. Friedman and J.W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9):881–890, 1974.
8. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Oswald and Rohatgi [14], pages 426–442.
9. Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications.* Elsevier, 2004.
10. Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *LNCS*, pages 222–235. Springer, 2004.
11. Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
12. Amir Moradi and François-Xavier Standaert. Moments-correlating DPA. *IACR Cryptology ePrint Archive*, 2014:409, 2014.
13. Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In Pointcheval [16], pages 192–207.
14. Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *LNCS*. Springer, 2008.
15. B. Poettering. Fast AES implementation for Atmel's AVR microcontrollers. http://point-at-infinity.org/avraes/.
16. David Pointcheval, editor. *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *LNCS*. Springer, 2006.
17. Emmanuel Prouff and Matthieu Rivain. A generic method for secure SBox implementation. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *WISA*, volume 4867 of *LNCS*, pages 227–244. Springer, 2007.
18. Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
19. Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *LNCS*, pages 109–128. Springer, 2011.
20. Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Selecting time samples for multivariate DPA attacks. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *LNCS*, pages 155–174. Springer, 2012.

21. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
22. Kai Schramm and Christof Paar. Higher order masking of the AES. In Pointcheval [16], pages 208–225.
23. François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Oswald and Rohatgi [14], pages 411–425.
24. François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *LNCS*, pages 253–267. Springer, 2008.
25. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.
26. François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *LNCS*, pages 112–129. Springer, 2010.