Robust Profiling for DPA-Style Attacks

Carolyn Whitnall and Elisabeth Oswald

University of Bristol, Department of Computer Science,
Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK.
{carolyn.whitnall, elisabeth.oswald}@bris.ac.uk

Abstract. Profiled side-channel attacks are understood to be powerful when applicable: in the best case when an adversary can comprehensively characterise the leakage, the resulting model leads to attacks requiring a minimal number of leakage traces for success. Such 'complete' leakage models are designed to capture the scale, location and shape of the profiling traces, so that any deviation between these and the attack traces potentially produces a mismatch which renders the model unfit for purpose. This severely limits the applicability of profiled attacks in practice and so poses an interesting research challenge: how can we design profiled distinguishers that can tolerate (some) differences between profiling and attack traces?

This submission is the first to tackle the problem head on: we propose distinguishers (utilising unsupervised machine learning methods, but also a 'down-to-earth' method combining mean traces and PCA) and evaluate their behaviour across an extensive set of distortions that we apply to representative trace data. Our results show that the profiled distinguishers are effective and robust to distortions to a surprising extent.

1 Introduction

1.1 Motivation

The aim of side-channel analysis is to discover—'learn'—information about the (secret) internal configuration of a cryptographic device from physical measurements (power consumption, electromagnetic radiation, run time, etc.) collected while the device is in operation. The discipline of machine learning is precisely concerned with computational algorithms which are able to 'learn' from data—discover patterns, arrive at meaningful generalisations, make predictions about previously unseen data instances, and so on. There is consequently a very natural overlap between the two fields, and increasing attention has been paid to the potential uses of machine learning techniques as tools for extracting information from side-channel measurements.

One pertinent problem when learning from data arises when the data is noisy and some characteristics change across data sets. Such changes are unfortunately to be expected in the context of side-channel attacks for at least two reasons. Firstly, an adversary is unlikely to have access to the precise target device during the learning phase and will be forced to make do with a duplicate device. Secondly, the measurement setup during the attack might not be the same as the lab setup used during the learning phase. We are hence interested, given the multitude of machine learning methods, which (if any), could be somewhat robust with regards to some practically meaningful disparities between the data sets used for profiling and the data set used during an attack.

1.2 Machine learning for profiling

The umbrella term 'machine learning' covers a variety of methods, which we categorise loosely as 'supervised' and 'unsupervised'. The former describes procedures which are provided (in a training phase) with a set of data instances and corresponding *a priori* known outputs (or 'labels'), and subsequently aim to generalise the relationship between the instances and the outputs in such a way as to reliably map new instances to their corresponding, otherwise unknown, outputs. The latter describes procedures which do not have access to *any* known outputs but seek to find patterns based on the inherent attributes of the instances relative to one another.

The earliest proposals that utilised some form of 'learned' characteristics from profiling data sets achieved this via Bayesian classification in a supervised manner: so called template DPA attacks utilise multivariate Gaussian distributions, which are built in a profiling phase [6] from traces with a known key. Recent strategies have incorporated more explicit machine learning tools such as support vector machines (SVM) [12,11,14] and random forests [14]. Theoretically, any supervised classification method could be chosen—with varying degrees of success as different algorithms are more or less suited to different underlying data structures. This is already a much-explored theme of recent research and we do not intend to extend it in this work.

We will focus rather on *unsupervised* techniques—in particular, unsupervised clustering algorithms. Clustering is the task of grouping objects (in this case, observed power consumption traces) in such a way that the objects inside any given group are *similar* to one another whilst objects in different groups are *dissimilar*. Unlike supervised classification, where new objects are assigned to an existing class based on knowledge of objects already within that class, a clustering algorithm aims to find a meaningful arrangement of objects with no *a priori* knowledge about the number or characteristics of the underlying classes.

1.3 Unsupervised clustering in conjunction with partition-based DPA

By applying an unsupervised clustering algorithm to leakage measurements with known sensitive values we thus learn a *meaningful partition* of the target values (a 'nominal power model' in the terminology of [25]). Our suggestion is to extract such a nominal power model in a profiling phase designed to be followed by a partition-based DPA attack [20] (mutual information (MI) [10], Kolmogorov-Smirnov (KS) [23], the variance ratio (VR) [20] and its multivariate extension in the context of Differential Cluster Analysis (DCA) [4], to name a few examples).

Such a strategy represents an interesting middle course between completely unprofiled attacks relying on difference-of-means or on 'typical' power models such as the Hamming weight (HW) (which in many cases—especially in attacks against hardware implementations—do not apply), and fully profiled attacks which comprehensively (and expensively) characterise entire multivariate distributions for leakage traces. A 'nominal' power model need not be perfect in order for a partition-based DPA to succeed; as long as it captures some (part of a) meaningful pattern then, provided there is enough data to estimate the distinguishing statistic sufficiently precisely, key recovery becomes feasible[1].

A key advantage of the suggested method is that it is potentially highly robust and portable: since no 'meaning' is ever attached to the cluster labels, there is less scope for their relevant interpretation to be disrupted by changes between the profiling and attack scenarios (e.g. measurement set-up, environmental conditions, device age, and imprecise location of interesting windows); all that is required for the power model to apply effectively to the attack measurements is that the arrangement of 'similarly leaking values' be preserved. This is *far* less stringent than requiring the characteristics and precise locations of conditional multivariate Gaussian distributions to be preserved, which is necessary in order to port Bayesian templates. The practical challenges of template attacks when the profiling and attack measurements are generated by distinct devices (or even just distinct acquisition campaigns) are the subject of considerable attention in the literature [18,8]; Choudhary et al. [7] find that the main difference is a DC offset, which may be compensated for to some extent by simply mean-centering the traces and/or via well-chosen compression techniques such as linear discriminant analysis or PCA. However, to our knowledge, none of the proposed methods are able to handle the type of *horizontal* misalignment or discrepancy between the profiling and attack measurements to which our (intentionally less precise) method is robust.

[1] Note that, in the case that the target function is injective (e.g. the AES S-box), the 'trivial' nominal power model which treats each intermediate value as a distinct class *invariably* fails to distinguish between key hypotheses in *any* partition-based DPA (see [20,25]). Therefore, a meaningful non-trivial grouping is required.

1.4 Our contributions

Firstly, we present a general strategy to integrate (unsupervised) clustering into a DPA attack flow, which is independent of the particular clustering algorithm and partition-based distinguisher selected. This is important because the effectiveness of machine learning tasks is notoriously sensitive to the choice of algorithm; the 'best' choice depends on the form of the data and is generally not known *a priori*.

Secondly, we present a couple of example realisations (using K-means and agglomerative hierarchical clustering, with the univariate [20] and multivariate [4] variance ratio as DPA distinguisher) and show that they do indeed succeed against a hardware as well as a software implementation of AES. We also propose an heuristic for extracting a *proportional* power model under identical profiling assumptions, for use in a subsequent correlation DPA. This approach outperforms the clustered profiling in the software setting (where the device leaks approximately the Hamming weight of the intermediates); naturally, clustered profiling maintains an advantage in the hardware setting (where the leakage of the device is a complex function).

Thirdly, we evaluate the distinguishers across a wide range of 'distortions' that we apply to our real world data. We find that the distinguishers remain effective in a wide range of scenarios where full templating is impossible (or in the best case very problematic), such as small profile samples, inaccuracy in identifying exact leakage points, and misalignment, varying measurement precision, and alternative pre-processing between the attack and profile samples.

1.5 Outline

The rest of the paper proceeds as follows. In Sect. 2.1 we overview DPA, with special attention to profiling and to the variance ratio as a DPA distinguisher; in Sect. 2.2 we overview unsupervised clustering in general, and K-means and hierarchical clustering in particular; in Sect. 2.3 we overview Principal Component Analysis. Then, in Sect. 3 we describe our general methodology for 'rough-and-ready' profiling and the subsequent attack phases. In Sect. 4 we present our experimental results, and we conclude in 5.

2 Preliminaries

2.1 Differential power analysis

We consider a 'standard DPA attack' scenario as defined in [16], and briefly explain the underlying idea as well as introduce the necessary terminology here. We assume that the power consumption $\mathbf{P} = \{P_1, ..., P_T\}$ of a cryptographic device (as measured at time points $\{1, ..., T\}$) depends, for at least some $\boldsymbol{\tau} \subset \{1, ..., T\}$, on some internal value (or state) $F_{k^*}(X)$ which we call the *target*: a function $F_{k^*} : \mathcal{X} \rightarrow \mathcal{Z}$ of some part of the known plaintext—a random variable $X \overset{R}{\in} \mathcal{X}$—which is dependent on some part of the secret key $k^* \in \mathcal{K}$. Consequently, we have that $P_t = L_t \circ F_{k^*}(X) + \varepsilon_t, t \in \boldsymbol{\tau}$, where $L_t : \mathcal{Z} \rightarrow \mathbb{R}$ describes the data-dependent leakage function at time t and ε_t comprises the remaining power consumption which can be modeled as independent random noise (this simplifying assumption is common in the literature—see, again, [16]). The attacker has N power measurements corresponding to encryptions of N known plaintexts $x_i \in \mathcal{X}$, $i = 1, \ldots, N$ and wishes to recover the secret key k^*. The attacker can accurately compute the internal values as they would be under each key hypothesis $\{F_k(x_i)\}_{i=1}^{N}$, $k \in \mathcal{K}$ and uses whatever information he possesses about the true leakage functions L_t to construct a prediction model (or models) $M_t : \mathcal{Z} \rightarrow \mathcal{M}_t$.

A distinguisher D is some function which can be applied to the measurements and the hypothesis-dependent predictions in order to quantify the correspondence between them, the intuition being that the predictions under a correct key guess should give more information about the true trace measurements than an incorrect guess. For a given such comparison statistic, D, the *theoretic* attack vector is $\mathbf{D} = \{D(L \circ F_{k^*}(X) + \varepsilon, M \circ F_k(X))\}_{k \in \mathcal{K}}$, and the *estimated* vector from a practical instantiation of the attack is $\hat{\mathbf{D}}_N = \{\hat{D}_N(L \circ F_{k^*}(\mathbf{x}) + \mathbf{e}, M \circ F_k(\mathbf{x}))\}_{k \in \mathcal{K}}$ (where $\mathbf{x} = \{x_i\}_{i=1}^N$ are the known inputs and $\mathbf{e} = \{e_i\}_{i=1}^N$ is the observed noise). Then the attack is *o-th order theoretically successful* if $\#\{k \in \mathcal{K} : \mathbf{D}[k^*] \leq \mathbf{D}[k]\} \leq o$ and *o-th order successful* if $\#\{k \in \mathcal{K} : \hat{\mathbf{D}}_N[k^*] \leq \hat{\mathbf{D}}_N[k]\} \leq o$.

Profiled DPA A profiled DPA attack is one in which the adversary has access to (and control of) a device matching the one they intend to target. They can therefore, in a preliminary stage, build informed models for the secret-value-dependent form of the device leakage [6,19,12]. The measurements obtained from a target device can then be compared with these models (e.g. using Bayesian classification) to reveal the most likely secret values. The motivation behind our clustering-based profiled DPA attack is to use an unsupervised clustering algorithm to obtain a meaningful mapping from intermediate values to leakage classes in a profiling phase, which can be used in a subsequent attack phase to hypothetically map new traces to classes under each key guess, thus revealing the secret key as the one associated with the most demonstrably 'meaningful' arrangement. Of course, we do not expect such a method to be anywhere near as *efficient* as a detailed, multivariate Gaussian template in the key recovery phase of the attack, but it is precisely its lack of detail and specificity which enables it to remain *effective* in non-ideal attack scenarios.

The variance ratio as a DPA distinguisher Because clustered profiling outputs a 'nominal' power model—a labelling of distinct leakage classes where the labels themselves are arbitrary—the DPA phase of the attack must use a distinguishing statistic which is invariant to re-labelling of those classes (as per [25]). These coincide with the 'partition-based' distinguishers identified by Standaert et al. in [20], and include the MI [10], the KS two sample test statistic [23], and the variance ratio [20].

We choose to practically verify our strategy using the latter of these, because of its conceptual simplicity, its computational efficiency, its good performance in previous studies [20,24], and the fact that it very naturally extends to multivariate DCA attacks as shown by Batina et al. in [4]. The variance ratio ranks hypothesis-dependent cluster arrangements according to the proportion of the overall variance which is accounted for:

$$D_{\mathrm{VR}}(k) = \frac{\sum\limits_{t \in \boldsymbol{\tau}'} \mathrm{var}(\{P_{t,i}\}_{i=1}^N)^2}{\frac{1}{N} \sum\limits_{m \in \mathcal{M}} n_m \sum\limits_{t \in \boldsymbol{\tau}'} \mathrm{var}(\{P_{t,i} | M \circ F_k(x_i) = m\})^2}, \tag{1}$$

where $\boldsymbol{\tau}'$ is the attacker's best knowledge about $\boldsymbol{\tau}$ (one hopes that $\boldsymbol{\tau}' \cap \boldsymbol{\tau} \neq \emptyset$), M is a nominal approximation (taking values in $\mathcal{M}$) for the leakage output by unsupervised cluster-based profiling, and $n_m = \#\{x_i | M \circ F_k(x_i) = m\}$, i.e. the number of observations in the trace set for which the predicted cluster label is m.[2]

2.2 Unsupervised clustering

Clustering is the task of grouping objects together so that those inside any given group are *similar* to one another whilst those in different groups are *dissimilar*, without any *a priori* knowledge about the number or characteristics of the underlying classes (unlike supervised classification). All methods learn through an iterative process involving trial and error, and vary widely in application and effectiveness depending on the assumed cluster model (hierarchical, centroid-based, density- or distribution-based, graph-based, and so on) the chosen 'similarity' measure (e.g. the Euclidean distance between objects), the thresholds chosen for inclusion or exclusion, and the conjectured number of clusters.

[2] The variances in Eqn. (1) are squared as per [4]; this makes the univariate VR slightly different to the original definition given in [20], but (importantly) consistent with the multivariate version.

***K*-means clustering** K-means clustering aims to partition the N data objects into K clusters such that each object belongs to the cluster with the nearest mean. The mean vectors are called the cluster *centroids*. Whilst conceptually simple, the actual arrangement is computationally difficult to achieve (NP-hard, in fact). Fortunately, heuristic algorithms exist which converge quickly to *local* optima over a series of refining iterations.

Formally, if $\{\mathbf{x}_i\}_{i=1}^{N}$ is a set of (real-valued) d-dimensional observations, the objective of K-means is to partition the N observations into $K < N$ clusters $\mathbf{C} = \{C_1, C_2, \dots, C_K\}$ so as to minimise the within-cluster sum of squares $\arg\min_{\mathbf{C}} \sum_{j=1}^{K} \sum_{\mathbf{x}_i \in C_j} ||\mathbf{x}_i - \mu_{\mathbf{j}}||^{\mathbf{2}}$.

'Lloyd's algorithm' is a popular heuristic solution:

1. Initialisation: Pick a set of K vectors to serve as the initial centroids (e.g. by choosing K observations at random from the dataset, by choosing K points uniformly at random from within the range of the dataset, or by computing the means of random clusters).
2. Assignment: Assign each observation to the "nearest" centroid, according to some appropriate distance metric (for example, the Euclidean, Manhattan or Hamming distance, or one minus the sample correlation between observations, depending on the type of data).
3. Update: If the assignments changed in step 2, calculate the means of the observations in the clusters and set these to be the new centroids; else, return.

For our experiments, we use the in-built Matlab command `kmeans`, which performs the above as a preliminary phase (which may or may not converge to a local minimum). It then treats the output as the starting point for an 'online' phase, in which points are reassigned *individually* (if doing so reduces the sum of distances), and the centroids recalculated after each reassignment (instead of in batch). This *will* converge to a local minimum, but it may not be a global minimum. Using several replicates with different random starting points can increase the likelihood of finding a solution which is a global minimum. We initialise the centroids by drawing K observations at random for each of 5 replicate runs, and we measure closeness according to the Euclidean distance.

Of course, since we are primarily interested in what can be achieved without prior information on the leakage, we suppose that the correct number of clusters K is unknown and must be discovered from the data as part of the machine learning task. We propose to search over different values of K and see which produces the 'best' clustering. Different notions of cluster quality exist; we choose to work with the *silhouette value*, defined for the i^{th} object as $S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$, where a_i is the average distance from the i^{th} object to the other objects in the same cluster, and b_i is the minimum (over all clusters) average distance from the i^{th} object to the objects in a different cluster. In our experiments, we select the number of clusters K to be the one producing the highest mean silhouette value.

Hierarchical clustering Hierarchical clustering arranges data objects into a multi-level tree of nested partitions. Clusters which are close on one level are joined at the next level, so that once objects are associated with each other they remain so at all higher levels of the tree. Strategies to achieve this can either be *agglomerative*, so that each observation starts in its own cluster, and clusters are merged as the tree is ascended, or they can be divisive, so that all observations start in one single cluster which is incrementally split as the tree is descended.

An agglomerative procedure proceeds as follows:

1. Compute pairwise 'dissimilarity' between objects: Typical notions of distance include Euclidean, Manhattan, Minkowski, Mahalanobis and Chebychev, but the algorithm is flexible to other dissimilarity measures which may or may not strictly satisfy the definition of a metric.
2. Initialise the clusters: We begin with N singleton clusters comprising the individual objects of the dataset.
3. While $K > 1$ (i.e., until all objects are collected together in a single cluster at the top of the tree):

  - Compute distance between clusters: Once there is more than one object in a cluster, there are different ways to do this, e.g., the shortest, furthest, or average distance between objects in two clusters.
  - Merge 'close' clusters in pairs.
4. Identify clusters: Partition objects according to the tree structure, either by computing the *inconsistency* associated with each link[3] and selecting those above a certain threshold, or by pruning the tree at the point corresponding to a fixed desired number of clusters.

Our experiments use the Matlab implementation of the above, with the Euclidean distance as the dissimilarity measure (step 1) and average cluster linking (step 3). For step 4, partitioning according to consistency thresholds should lead to the 'most natural' arrangement and number of clusters; because of the difficulty of *a priori* selecting the appropriate consistency threshold, we tested all values from 0.9 to 1.2 in increments of 0.02 and, as for the K-means clustering, choose the one producing the largest silhouette index.

2.3 Principal component analysis

Principal component analysis (PCA) is a popular method for dimensionality reduction. An $n \times m$ matrix is orthogonally transformed so that the m columns in the new matrix are linearly uncorrelated and sorted in decreasing order of variance. By construction, the columns are the eigenvectors of the covariance matrix, sorted according to the size (largest to smallest) of the corresponding eigenvalues $\lambda_1, \ldots, \lambda_m$. The first $q < m$ of these columns maximise (w.r.t. all other $n \times q$ transformations) the total variance preserved whilst minimising the mean squared reconstruction error $\sum_{i=q+1}^{m} \lambda_i$. The hope is that all of the 'important' information will be concentrated into a small number of components.

PCA has been proposed as a means of locating 'points of interest' for inclusion in Gaussian templates [2,17]. It has also been used to pre-process traces for more efficient non-profiled correlation DPA attacks [5]. Moreover, it is typically used in combination with unsupervised clustering algorithms to concentrate the relevant information into a lower dimensional data space so as to avoid the problem of sparseness, where *no* observations are 'close' (sometimes called the 'curse of dimensionality'). It is natural, then, for us to transform the trace data so as to work with high-ranked principal components only in the clustering phase. The precise number to retain is normally determined by (arbitrarily) setting a threshold for the proportion of total variance explained, but since our goal is to find the 'best' cluster arrangement we select the number of projected dimensions (up to 10) depending on the silhouette values attained in each case.

3 Methodology

The profiling strategy we suggest is independent of the specific choice of learning algorithm: it can operate with any clustering technique $\mathcal{C}$ which returns a mapping M from intermediate values of the algorithm $z \in \mathcal{Z}$ to a set of nominal cluster labels $m \in \mathcal{M}$. The success of the subsequent attack stage depends, of course, on the validity of the clusters discovered by the learning algorithm[4].

3.1 Our general profiling strategy

Let $\{\mathbf{t_1}, \ldots, \mathbf{t_{N_p}}\}$ be a set of $1 \times T$ trace measurements taken from a profiling device sufficiently similar to the target. Let $\{z_i\}_{i=1}^{N_p}$ be a set of known intermediate values handled by the device during the interval spanned by the measurements. The strategy, in its most general form, is as follows:

[3] Defined as the height of the individual link minus the mean height of all links at the same hierarchical level, all divided by the standard deviation of all the heights on that level.

[4] In particular, in the notation of Sect. 2.1, the extent to which $\{z' | M(z') = M(z)\} \approx \{z' | L(z') = L(z)\} \forall z \in \mathcal{Z}$—see [25].

1. Partition the data according to the intermediate values and compute the mean traces $\{\bar{\mathbf{t}}_z\}_{z\in\mathcal{Z}}$.
2. Obtain a mapping $M : \mathcal{Z} \longrightarrow \mathcal{M}$ by clustering the mean traces. Values in $\mathcal{Z}$ not represented in the profiling dataset are mapped to cluster $C+1$ where C is the total number of clusters identified by the chosen algorithm (essentially, an 'other' category).
3. Use M as the power model in 'partition-based' DPA.

3.2 Model building and distinguishers

In practice, there are many options open to the attacker in steps 2 and 3. It is notoriously difficult to *a priori* apply the most well-suited machine learning solution to any particular problem instance [26], and an exhaustive testing of all possible strategies is infeasible. Given the infeasibility to find 'optimal' strategies across scenarios, we provide some meaningful choices using the methods outlined in Sect. 2.2 with varying parameters. We suggest to first perform PCA on the mean traces[5] and then to experimentally obtain the best clusterings we can via each of the two algorithms. We vary the number of components retained as well as a) the specified number of clusters for the K-means algorithm, or b) the consistency threshold for the agglomerative hierarchical algorithm. In both cases the 'best' cluster arrangements are identified according to the silhouette index. This 'best' model is the one used for the DPA attack, which (for the purposes of verifying feasibility) we perform using the variance ratio for its conceptual and computational simplicity, and its natural multivariate extension DCA from [4]. For the univariate variant (denoted VR(M), where M is either the K-means acquired (M_{KM}) or the hierarchical clustering acquired (M_{HC}) power model) we compute Eqn. (1) pointwise across the window and select the (key guess,time point) pair which produces the largest score (see Eqn. (2) below); for the multivariate variant (DCA(M)) we compute Eqn. (1) for the entire window in one go (see Eqn. (3)).

$$D_{\mathrm{VR}(M)}(k) = \max_{t\in\boldsymbol{\tau}'} \left\{ \frac{\mathrm{var}(\{P_{t,i}\}_{i=1}^{N})^2}{\frac{1}{N}\sum\limits_{m\in\mathcal{M}} n_m \mathrm{var}(\{P_{t,i}|M\circ F_k(x_i)=m\})^2} \right\}, \tag{2}$$

$$D_{\mathrm{DCA}(M)}(k) = \frac{\sum\limits_{t\in\boldsymbol{\tau}'} \mathrm{var}(\{P_{t,i}\}_{i=1}^{N})^2}{\frac{1}{N}\sum\limits_{m\in\mathcal{M}} n_m \sum\limits_{t\in\boldsymbol{\tau}'} \mathrm{var}(\{P_{t,i}|M\circ F_k(x_i)=m\})^2}, \tag{3}$$

where M is either the K-means acquired (M_{KM}) or the hierarchical clustering acquired (M_{HC}) power model.

We also (by way of comparison) introduce a counterpart heuristic to 'profile' for correlation DPA on a similar basis. Firstly (denoted M_{P1}), we use the projection of the mean traces along the first principal direction as the power model; secondly (denoted M_{P2}), we take all the projections accounting for 70% of the variance in the mean traces, weight them by their contribution, and either add or subtract them from a running total depending on their positive or negative correlation with the first principal direction (thus allowing for the possibility that the relevant variation is contained in more than one component). Analogous to the nominal profiling, values in $\mathcal{Z}$ which are not represented in the sample are mapped to the global mean. We exploit these power models by computing the univariate correlation distinguishing vectors at each point in time in the attacked traces, and choosing the (key guess,time point) pair producing the highest score (see Eqn. (4) below). Correlation DPA seems a fitting benchmark because of its known good performance, but we certainly do not make any claims about the optimality of our 'profiling' methods—they are merely heuristics to produce power models under the same restrictions as the clustering analyses.

$$D_{\mathrm{Corr}(M)}(k) = \max_{t\in\boldsymbol{\tau}'} \left\{ \frac{\mathrm{cov}(\{P_{t,i}\}_{i=1}^{N}, M\circ F_k(\mathbf{x}))}{\sqrt{\mathrm{var}(\{P_{t,i}\}_{i=1}^{N})\mathrm{var}(M\circ F_k(\mathbf{x}))}} \right\}, \tag{4}$$

[5] Note that this process involves centering around the global mean, thereby avoiding the DC offset problems highlighted by [7].

where M is the proportional model acquired either from the first principal direction (M_{P1}) or by combining information from the directions accounting for 70% of the variance (M_{P2}).

3.3 Experimentally verifying 'robustness'

Different measurement set-ups, pre-processing and device ageing introduce discrepancies between the profiling and attack samples. DC offset has been recognised as a significant obstacle to classical Gaussian templating which can be overcome by appropriate compression and normalisation [7]. Since our method naturally incorporates these steps, it is also robust to DC offset. However, it goes much further: it operates on the raw attack-stage traces, without requiring to know or apply the principal subspace projection derived and applied in the profiling stage, nor even the precise points or window of points for which the profiled models were built. Hence acquisitions from the target device need not be made at the same frequency, nor subjected to the same filtering or compression techniques, for the attack to be implemented. We test the effectiveness of our method against the following practically relevant scenarios[6]:

- The precise width and location of the window of points used to build the cluster-based power model is not known in the attack phase. We simulated this scenario by choosing non-matching windows. (See Sect. 4.3).
- The attack traces are measured at a different resolution to those from which the template is built. We achieved this by binning trace values with increasing coarseness. (See Sect. 4.4).
- The attack traces contain more measurement error. We achieved this by adding Gaussian noise in increasingly large proportion to the observed conditional noise. Note that this incorporates the scenario in which the traces are misaligned (possibly deliberately, via 'hiding in the time dimension' [15]; it also covers countermeasures such as [9,3] which are based on frequency/voltage changing. Assuming *some* proportion of the traces coincide for a given intermediate value, the signal will persist weakly, with the remaining (non-aligned) traces functioning as noise. (See Sect. 4.5).
- The attack traces have been differently pre-processed. We achieved this by taking a moving average of increasing window width. (See Sect. 4.6).
- The attack traces are imperfectly aligned, as though (for example) the dynamic power saving technique of [9] had been in operation, or a hiding countermeasure such as [27]. Whilst methods exist to improve alignment (see, e.g. [22]), none are known to remove the problem entirely. By ranging from small to greater distortions, we approximate cases in which alignment methods have been applied with varying success. We achieved this by inserting an increasing number of 'interpolated' values in random positions in each trace. (See Sect. 4.7).

4 Experimental results

We test our strategies on leakages acquired from two unprotected implementations of AES—one software, running on an ARM microcontroller (10,000 traces total); one hardware, designed for an RFID-type system (5,000 traces total)[7]. In each case, we perform repeated experiments on random subsamples of the data, for increasing profiling and (disjoint) attack sample sizes with a fixed window width (20 for the software traces, 10 for the hardware, because of the coarser

[6] All our data stems from *real devices*: one implementation of AES on an ARM7 processor, and one implementation of AES in dedicated hardware (an ASIC custom-built for the TAMPRES project [1,13]) using a 32-bit architecture but with a serial S-box look-up. In order to create data sets with different characteristics we did however not change the measurement setups as this would have been a too cumbersome process. Instead we manipulated the original data sets and hence, strictly speaking, the distorted data was created by simulations.

[7] The different sample sizes reflect the fact that we sourced independently-generated datasets for our experiments rather than relying on acquisition set-ups over which we had full control.

granularity of the latter) around the (already identified) 'most interesting' point.[8] We then explore the robustness of the attacks to different window widths and to the various profiling/attack trace discrepancies detailed above.

4.1 'Straightforward' (software) scenario

Fig. 1 shows the guessing entropies (average ranks of the correct subkey [21]) after attacks against the output of the first S-box in software as the sample sizes vary. Crucially, the clustering strategy can be seen to 'work'—that is, *all* the variants reduce uncertainty about the subkey. The K-means clustering (denoted '(M_{KM})' in the legend) appears to require a larger profiling sample to produce an effective power model than the hierarchical clustering (denoted '(M_{HC})'), but eventually outperforms the latter. The multivariate VR distinguisher (aka DCA [4]) outperforms the univariate one in the case of both clustered profiles. In this 'straightfoward' scenario (the leakage is known to correspond closely to the HW) our heuristics for acquiring proportional power models also prove effective so that both correlation attacks (denoted 'Corr(M_{P1})' and 'Corr(M_{P2})') outperform all those using clustered profiling, with slight advantage to the one relying only on the first principal direction (M_{P1}). These are even able to recover the subkey within 800 attack measurements from a profiling dataset of just 200.

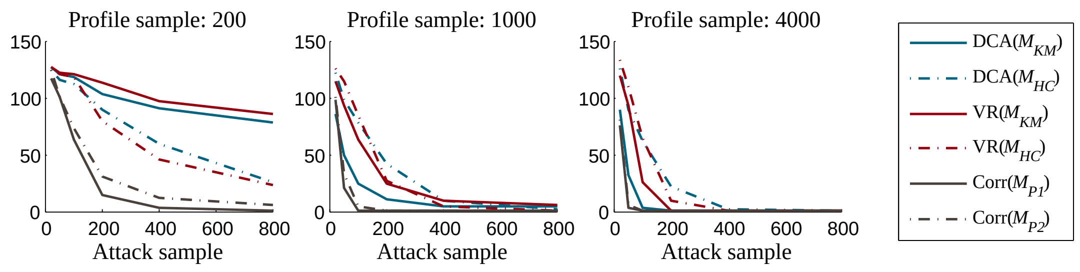

Fig. 1: *Guessing entropy of partially profiled DPA attacks against an unprotected software implementation of AES. Window width: 20; reps: 500.*

4.2 'Problematic' (hardware) scenario

Hardware leakages are typically less 'easy' to exploit (e.g. in simple attacks using the HW power model; indeed, we tested and found such attacks to fail to recover the key even when provided with the full 5,000 measurements). The implementation that we target has two working 32-bit registers, with the byte substitutions in each column occurring in parallel with the MixColumns operation on the previous column. This makes it much harder to isolate a single contributory process in the overall leakage.

Preliminary investigations revealed considerable variation in the exploitability of the different S-boxes; we picked one (S-box 14) which was more amenable to attack in order to report interesting (but clearly not definitive) results (see Fig. 2). In this case, the K-means-based profiling coupled with the (multivariate) DCA distinguisher performs particularly strongly, even outperforming the best of the two correlation attacks (especially when only 200 traces are available for profiling).

We thus learn that there may be cases where 'cheap' rough-and-ready nominal profiling, with minimal prior knowledge, is a relatively effective option. However, our results are by no means conclusive: performance of machine learning methods is notoriously scenario-dependent [26], so that (e.g.) minor alterations in the chosen location or width of the trace window, as well as different

[8] Identified by using the (point-wise) conditional means as optimal power models in (point-wise) correlation DPA, and selecting the one giving the strongest margin of success.

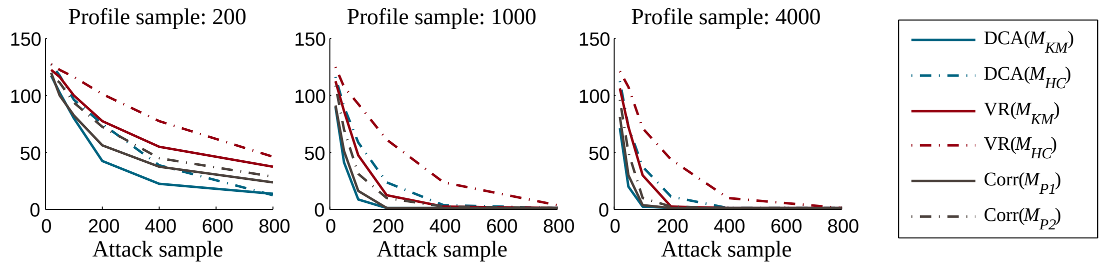

Fig. 2: *Guessing entropy of partially profiled DPA attacks against an unprotected hardware implementation of AES. Window width: 10; reps: 500.*

algorithms or parameters, may produce wildly different outcomes. (We test for this in the next section). Moreover, we have not here strived towards a 'best' method for acquiring a proportional power model to act as a definitive benchmark. Nonetheless, we consider these experimental results to be an interesting insight into what is possible.

Tab. 1 summarises the parameters chosen by our cluster model selection rule. As the software profiling sample size increases, the number of clusters (on average) detected by K-means also increases, close to 9 (unsurprisingly—it is known to closely follow the HW leakage function). The mean is around 7 or 8 for all sample sizes in the case of the hardware leakage (where less is known *a priori*); around 5 principal components are retained in both cases. The hierarchical algorithm finds quite different (less concise) arrangements, almost always using only one principal component. However, it clearly captures *something* meaningful about the true arrangement of the target values, as the effective (though inefficient) attack phases confirm.

	Software				**Hardware**			
	K-means		**Hier.**		**K-means**		**Hier.**	
Sample size	K	**#PC**	K	**#PC**	K	**#PC**	K	**#PC**
200	2.1	5.2	55.8	1.0	6.5	5.3	50.1	1.0
1000	4.3	5.3	93.6	1.0	7.9	5.2	92.6	1.0
4000	8.6	5.2	94.1	1.0	8.4	5.3	94.6	1.0

Table 1: *Summary of selected cluster-based power models. Window width: 20 for the software, 10 for the hardware; reps: 500. Table reports means.*

4.3 Discrepancy in window width and location

We now consider scenarios in which the trace window used to derive the power models varies, and in which it differs (in width and/or location) to the window selected for the attack.

Tabs. 2 and 3 show the guessing entropies attained by the attacks against the software traces for different trace window widths, for attack sample sizes of 50 and 400. We fixed the profiling sample size at 4000 and the attack sample size at 50/400 whilst varying both the profiling and attack window widths, keeping the known interesting point central.

All six attacks are robust to mismatch between the windows, and remain effective at reducing the subkey guessing entropy as the widths vary; however there is some cost to attack efficiency (in terms of the number of traces needed) as the windows become less specific. All of the power models become less effective as the profile window widens and, to a lesser extend, as the attack window widens. The correlation power models appear the most robust; DCA with a hierarchical model the least (especially with regard to the attack sample, which we might expect as all the points in a window are included to compute a single distinguisher value under each subkey hypothesis).

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
width ⟶		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	3	2	3	5	17	3	9	13	26	32	1	1	1	2	6
	4	3	2	6	14	27	3	12	34	40	49	1	1	3	8	13
	10	3	4	6	10	31	3	10	25	51	61	1	3	4	7	9
	20	27	18	24	37	60	27	53	63	89	102	1	1	1	2	4
	40	72	70	76	84	95	72	94	104	106	102	1	3	6	8	7
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
width ⟶		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	61	70	84	101	113	61	92	101	104	116	1	1	1	2	6
	4	64	64	78	96	107	64	79	89	105	123	1	1	3	8	13
	10	49	55	73	83	100	49	81	98	110	118	1	3	3	3	11
	20	61	66	79	98	111	61	85	114	121	122	1	4	5	6	7
	40	89	86	100	107	123	89	100	113	113	115	1	3	12	20	26

Table 2: *Outcomes as the profiling and attack window widths vary: software implementation. (100 repetitions; profiling sample of 4000, attack sample of 50)).*

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
width ⟶		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	40	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
width ⟶		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	1	1	1	3	26	1	1	1	1	1	1	1	1	1	1
	4	1	2	2	3	17	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	2	11	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	4	20	1	1	1	1	1	1	1	1	1	1
	40	6	3	5	15	53	6	3	3	3	3	1	1	1	1	1

Table 3: *Outcomes as the profiling and attack window widths vary: software implementation. (100 repetitions; profiling sample of 4000, attack sample of 400)).*

Tabs. 4 and 5 present the counterpart outcomes against the hardware traces. Interestingly, the cluster-based profiling is very resilient to changes in the profile window width; the (univariate) VR distinguisher which uses them is moreover robust as the attack window widens, whilst the (multivariate) DCA distinguisher is penalised considerably for wider attack windows. By striking contrast, the two proportional power models quickly degrade as the *profiling* sample width increases, and are robust to changes in the *attack* sample width.

Tab. 6 shows the results of varying the *location* of the attack window relative to the profiling window in attacks against the software implementation, for a fixed window width of 20 and a fixed profiling sample size of 4000 (to ensure that the models themselves are well fitted). All tested attacks remain effective at reducing the key guessing entropy, though the number of traces required to achieve a comparable reduction does increase with the size of the offset (in either direction), especially in the case of hierarchical clustering.

The (univariate) VR and the correlation attacks against the hardware traces (Tab. 7) appear more robust to the offset; they suffer some penalty once the attack window is shifted forward by half the total window width, but at all other tested positions they do not even exhibit a reduction in efficiency. The (multivariate) DCA appear vulnerable to an equivalently large shift in the opposite direction, and in general vary more as they depart from the profile window position.

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
width $\longrightarrow$		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	12	30	43	119	135	12	64	83	88	99	2	11	22	36	42
	4	19	14	20	110	126	19	34	57	70	94	3	14	48	61	65
	10	50	37	24	117	120	50	61	72	92	93	4	24	40	46	64
	20	51	32	25	107	112	51	58	80	93	105	113	117	116	102	118
	40	49	42	35	113	123	49	70	79	95	105	126	127	121	134	136
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
width $\longrightarrow$		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	80	98	105	114	113	80	109	119	120	119	2	11	22	36	42
	4	95	87	95	112	119	95	93	112	118	124	17	13	36	53	57
	10	91	84	73	121	121	91	99	103	107	118	33	48	41	53	78
	20	84	90	88	124	121	84	99	107	111	109	78	81	90	98	119
	40	83	83	79	114	114	83	102	110	118	119	111	106	111	121	134

Table 4: *Outcomes as the profiling and attack window widths vary: hardware implementation. (100 repetitions; profiling sample of 4000, attack sample of 50)).*

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
width $\longrightarrow$		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	1	1	1	85	132	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	47	116	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	60	113	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	54	107	1	1	1	1	1	67	80	69	71	74
	40	1	1	1	68	119	1	1	1	1	1	126	118	109	118	123
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
width $\longrightarrow$		**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**	**1**	**4**	**10**	**20**	**40**
Profile width	**1**	4	22	34	113	117	4	23	28	28	28	1	1	1	1	1
	4	5	3	8	127	131	5	10	39	41	41	1	1	1	1	1
	10	10	3	1	100	117	10	13	9	9	9	1	3	1	1	1
	20	8	10	3	114	132	8	21	13	13	13	33	37	33	40	45
	40	14	6	3	109	120	14	27	25	25	25	78	89	91	95	103

Table 5: *Outcomes as the profiling and attack window widths vary: hardware implementation. (100 repetitions; profiling sample of 4000, attack sample of 400)).*

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Sample offset	**-10**	112	53	13	2	1	137	87	43	3	1	95	15	1	1	1
	-5	100	37	5	1	1	125	65	22	2	1	85	3	1	1	1
	0	88	34	4	1	1	117	72	25	2	1	87	1	1	1	1
	5	86	27	3	1	1	116	83	23	2	1	80	1	1	1	1
	10	107	74	31	9	4	122	109	60	9	1	111	22	1	1	1
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Sample offset	**-10**	121	104	101	57	18	130	118	81	11	1	85	16	1	1	1
	-5	116	97	86	34	7	126	113	65	9	1	87	7	1	1	1
	0	117	88	66	24	2	122	112	64	9	1	79	10	1	1	1
	5	118	89	63	22	2	122	113	62	15	1	85	8	1	1	1
	10	124	113	109	75	38	126	123	93	36	2	115	27	3	1	1

Table 6: *Outcomes when the attack sample window is misaligned with the profiling window: software implementation. (100 repetitions; window width of 20; profiling sample of 4000).*

Attack		**DCA(M_{KM})**					**VR(M_{KM})**					**Corr(M_{P1})**				
sample ⟶		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Sample offset	**-5**	121	97	84	65	37	68	28	3	1	1	22	8	1	1	1
	-2	51	9	1	1	1	66	20	3	1	1	20	5	1	1	1
	0	15	1	1	1	1	65	18	1	1	1	21	6	1	1	1
	2	25	3	1	1	1	76	19	3	1	1	24	6	1	1	1
	5	66	22	5	1	1	113	70	19	3	1	90	52	8	1	1
Attack		**DCA(M_{HC})**					**VR(M_{HC})**					**Corr(M_{P2})**				
sample ⟶		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Sample offset	**-5**	117	117	119	109	89	94	77	36	8	1	51	18	3	1	1
	-2	85	68	40	8	1	103	73	36	10	1	48	11	3	1	1
	0	65	32	10	1	1	98	68	36	9	1	56	10	3	1	1
	2	74	42	17	2	1	105	82	36	14	1	56	19	1	1	1
	5	93	75	55	28	4	117	105	77	35	8	91	68	38	10	7

Table 7: *Outcomes when the attack sample window is misaligned with the profiling window: hardware implementation. (100 repetitions; window width of 10; profiling sample of 4000).*

4.4 Discrepancy in measurement resolution

We next simulate discrepancy in measurement resolution, by discretising the attack sample measurements into fewer numbers of equally-sized bins. Tab. 8 shows the subsequent outcomes against the software implementation: the effectiveness of the attacks is unchanged, though for 32 bins or fewer there appears some penalty in terms of efficiency. This affects the correlation-based strategies as well as the clustered profiling.

Attack		**DCA(M_{KM})**					**VR(M_{KM})**					**Corr(M_{P1})**				
sample ⟶		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Number of bins	**256**	78	30	7	1	1	122	86	29	1	1	81	5	1	1	1
	128	81	28	7	1	1	128	83	32	1	1	81	5	1	1	1
	64	85	38	8	1	1	104	81	46	1	1	96	9	1	1	1
	32	104	68	21	1	1	131	107	57	13	1	91	29	1	1	1
	16	103	70	24	2	1	125	135	131	109	133	73	26	1	1	1
Attack		**DCA(M_{HC})**					**VR(M_{HC})**					**Corr(M_{P2})**				
sample ⟶		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Number of bins	**256**	118	86	58	27	3	122	107	82	10	1	72	3	1	1	1
	128	119	89	61	28	3	124	102	76	10	1	75	1	1	1	1
	64	116	91	75	33	5	136	93	70	18	1	83	8	2	1	1
	32	125	109	97	63	24	166	113	92	28	8	85	22	3	1	1
	16	124	112	100	71	31	176	132	177	119	93	80	31	1	1	1

Table 8: *Outcomes when the attack acquisition is measured with less precision than the profiling sample: software implementation. (100 repetitions; window width of 20; profiling sample of 4000).*

Tab. 9 shows the outcomes of the same offset attacks against the hardware traces. In contrast to the software scenario, the impact on efficiency (with respect to attack sample size) appears very minimal across most of the tested range, with some penalty for the most coarsely-binned traces, especially in the case of the hierarchical clusters.

4.5 Discrepancy in measurement error

Increased measurement error can be simulated simply by adding, to the raw traces, a (zero mean) Gaussian distributed random sample. The variance is chosen in increasing proportion to the (time

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Number of bins	**256**	16	1	1	1	1	68	18	1	1	1	23	6	1	1	1
	128	16	1	1	1	1	66	19	3	1	1	21	8	1	1	1
	64	17	2	1	1	1	62	21	2	1	1	29	10	1	1	1
	32	20	2	1	1	1	65	17	2	1	1	32	8	1	1	1
	16	33	6	1	1	1	71	37	6	1	1	55	8	1	1	1
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Number of bins	**256**	65	32	10	1	1	105	67	35	9	1	58	12	3	1	1
	128	64	32	11	1	1	101	65	37	9	1	56	9	4	1	1
	64	64	32	10	1	1	96	72	40	8	1	50	9	3	1	1
	32	67	36	11	2	1	89	79	44	9	1	59	9	3	1	1
	16	80	51	22	3	1	97	86	52	21	3	69	27	5	1	1

Table 9: *Outcomes when the attack acquisition is measured with less precision than the profiling sample: hardware implementation. (100 repetitions; window width of 10; profiling sample of 4000).*

point specific) conditional variance of the raw traces, computed via the residuals (the mean for all traces sharing the same intermediate value, subtracted from the raw measurement). For example (because of the additive properties of variance) to double the total conditional variance, one adds a sample with the same variance again; to triple it, one adds a sample with twice the variance, and so on.

Noise has the expected effect on all tested strategies (Tabs. 10 and 11): they remain effective, but the number of traces required for equivalent success scales proportionally. This is by contrast with, for example, strategies using Gaussian templates with Bayesian likelihood key recovery, which suppose that the random as well as the deterministic parts of the profiled leakage distributions match those of the attack-stage measurements.

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Noise factor	**1**	89	31	4	1	1	122	93	37	1	1	71	9	1	1	1
	2	107	71	20	2	1	128	103	75	8	1	92	33	1	1	1
	4	129	100	61	19	3	128	118	97	50	8	112	78	25	1	1
	8	126	124	98	50	14	133	115	115	93	38	116	103	76	18	1
	16	132	115	92	86	52	125	133	121	106	107	122	129	109	67	14
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Noise factor	**1**	112	87	62	26	3	121	106	68	12	1	68	14	1	1	1
	2	119	106	91	69	25	126	120	112	58	4	87	33	2	1	1
	4	126	114	108	100	66	128	136	124	89	53	114	76	33	1	1
	8	120	117	112	114	97	133	117	109	122	96	118	98	80	16	1
	16	119	124	109	125	116	129	124	116	130	119	121	127	103	75	18

Table 10: *Outcomes when noise in the attack sample increases relative to the profiling sample: hardware implementation. (100 repetitions; window width of 20; profiling sample of 4000)*

4.6 Discrepancy in trace pre-processing

It is straightforward to apply additional filtering to the attack traces; we do so by computing moving averages within a window of increasing width. Tabs. 12 and 13 show the outcomes of the attacks as the smoothing window widens; against the software implementation smoothing over two

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Noise factor	**1**	22	1	1	1	1	86	21	2	1	1	29	6	1	1	1
	2	56	11	2	1	1	107	62	16	1	1	65	17	3	1	1
	4	71	34	5	1	1	100	80	63	14	2	80	56	24	2	1
	8	116	67	32	7	1	123	112	95	50	15	95	86	44	9	5
	16	112	95	71	40	9	113	116	102	85	52	114	98	100	67	26
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Noise factor	**1**	75	41	15	1	1	95	81	45	4	1	45	11	1	1	1
	2	106	70	42	7	1	124	96	85	30	4	74	52	10	1	1
	4	110	93	71	34	8	118	117	109	71	35	93	76	26	12	1
	8	123	110	103	83	31	128	120	127	96	70	118	91	67	44	6
	16	123	116	111	101	76	125	126	119	126	100	126	117	99	77	36

Table 11: *Outcomes when noise in the attack sample increases relative to the profiling sample: software implementation. (100 repetitions; window width of 10; profiling sample of 4000)*

observations actually appears to *improve* attack outcomes in all cases. After that point outcomes degrade gradually. Attacks against hardware traces are slightly less robust, as we might expect, since the implementation completes in fewer clock cycles thereby giving rise to already shorter, more coarsely sampled traces. As before, the (generally more efficient) correlation variant is also robust to this particular discrepancy.

Attack		**DCA**(M_{KM})					**VR**(M_{KM})					**Corr**(M_{P1})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Smoothing window	**1**	89	43	7	1	1	119	96	29	1	1	69	16	1	1	1
	2	86	44	6	1	1	115	75	24	1	1	48	5	1	1	1
	4	93	51	8	1	1	126	104	24	1	1	45	5	1	1	1
	8	113	77	29	3	1	132	106	46	6	1	74	16	3	1	1
	16	131	115	78	38	5	124	123	95	45	3	87	53	16	1	1
Attack		**DCA**(M_{HC})					**VR**(M_{HC})					**Corr**(M_{P2})				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Smoothing window	**1**	137	77	53	26	2	147	110	71	7	1	71	12	1	1	1
	2	135	73	45	18	1	138	104	43	7	1	59	7	1	1	1
	4	135	73	52	23	1	131	99	52	10	1	53	8	1	1	1
	8	134	83	68	48	9	132	107	73	31	3	74	23	3	1	1
	16	139	107	96	95	54	138	110	104	84	36	89	55	18	2	1

Table 12: *Outcomes when the attack acquisition is smoothed via a moving average of increasing window width: software implementation. (100 repetitions; window width of 20; profiling sample of 4000)*

4.7 Non-fixed sampling frequency

Next we explore what happens to the attack outcomes when the traces are misaligned in some way which the attacker was unable to fully 'undo'—candidate causal scenarios include the power saving strategy proposed in [9] and the related DPA hiding countermeasure in [27] (though successfully circumvented in [3]). We simulate this distortion in our (already filtered) trace dataset by 'padding' an increasing proportion of sample points with simply interpolated additional values[9] in random positions which vary by trace.

[9] Computed as the mean of the preceding and following measurements.

Attack		**DCA(M_{KM})**					**VR(M_{KM})**					**Corr(M_{P1})**				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Smoothing window	**1**	19	1	1	1	1	62	25	1	1	1	19	3	1	1	1
	2	24	2	1	1	1	59	13	3	1	1	17	1	1	1	1
	4	74	36	9	1	1	100	74	28	4	1	79	42	5	1	1
	8	111	93	68	32	11	121	93	88	54	28	100	79	46	17	3
	16	112	113	104	82	58	118	113	100	94	71	113	112	90	64	53
Attack		**DCA(M_{HC})**					**VR(M_{HC})**					**Corr(M_{P2})**				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Smoothing window	**1**	78	42	11	1	1	99	81	45	10	1	42	13	1	1	1
	2	84	47	18	3	1	109	68	37	3	1	40	13	1	1	1
	4	114	90	72	51	11	113	103	79	44	9	84	60	23	14	5
	8	133	114	110	108	90	122	120	104	117	99	98	91	78	29	4
	16	125	115	108	116	107	125	120	118	113	110	127	124	108	95	55

Table 13: *Outcomes when the attack acquisition is smoothed via a moving average of increasing window width: hardware implementation. (100 repetitions; window width of 10; profiling sample of 4000)*

Table 14 shows outcomes of the attacks against the software implementation, which fail completely to reduce uncertainty on the correct subkeys. Table 15 shows outcomes against the hardware, in which case, for very small amounts of distortion the correlation and k-means variance ratio attacks seem to have some extremely success in reducing uncertainty as the number of traces increases.

Attack		**DCA(M_{KM})**					**VR(M_{KM})**					**Corr(M_{P1})**				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Insertions (prop.)	**0.005**	141	133	121	116	125	138	131	127	125	124	127	139	142	140	137
	0.01	136	126	121	116	111	125	134	121	131	119	132	128	139	119	135
	0.05	131	120	119	124	135	129	133	123	113	123	120	131	128	145	123
	0.1	134	141	129	129	134	129	131	127	124	127	132	129	129	136	134
	0.5	140	130	115	104	113	129	138	135	123	121	141	116	120	126	131
Attack		**DCA(M_{HC})**					**VR(M_{HC})**					**Corr(M_{P2})**				
sample $\longrightarrow$		**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**	**20**	**50**	**100**	**200**	**400**
Insertions (prop.)	**0.005**	141	132	127	126	115	136	132	115	131	118	133	137	127	132	138
	0.01	135	129	130	127	122	139	126	131	134	132	129	120	123	126	144
	0.05	138	134	137	130	126	136	129	133	132	121	130	127	135	137	132
	0.1	135	133	138	131	119	138	116	129	126	126	136	120	129	120	127
	0.5	135	131	135	132	122	131	133	120	145	123	129	109	122	131	124

Table 14: *Outcomes when the attack traces are misaligned, as the proportion of sample points padded increases: software implementation. (100 repetitions; window width of 20; profiling sample of 4000)*

5 Summary

We have shown that unsupervised clustering can recover nominal power models for use in effective 'partition-based' key recovery attacks, with minimal requirements in the profiling phase and a degree of flexibility in the attack phase, particularly when it comes to distorted attack traces. Via DCA they present a naturally multivariate methodology to exploit multiple trace points without requiring alignment, identical measurement set-up, or equivalent pre-processing between the profile

Attack		**DCA(M_{KM})**					**VR(M_{KM})**					**Corr(M_{P1})**				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Insertions (prop.)	**0.005**	122	127	125	125	130	122	111	123	97	73	117	90	78	46	7
	0.01	135	139	130	127	127	123	132	137	146	160	139	112	122	108	102
	0.05	125	130	135	117	114	126	130	120	127	118	125	125	131	131	125
	0.1	131	130	125	116	119	138	128	114	135	125	126	136	113	135	130
	0.5	143	135	124	131	128	128	118	128	138	122	134	141	123	131	127
Attack		**DCA(M_{HC})**					**VR(M_{HC})**					**Corr(M_{P2})**				
sample $\longrightarrow$		**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**	**50**	**100**	**200**	**400**	**800**
Insertions (prop.)	**0.005**	115	120	130	144	124	110	129	126	111	110	113	111	80	50	29
	0.01	121	134	140	143	139	120	130	123	116	138	128	118	124	105	99
	0.05	127	127	123	118	126	126	128	121	126	129	131	123	129	127	134
	0.1	119	113	126	126	132	111	121	127	128	122	117	122	120	128	134
	0.5	134	132	130	123	124	138	129	123	113	132	133	118	135	121	128

Table 15: *Outcomes when the attack traces are misaligned, as the proportion of sample points padded increases: hardware implementation. (100 repetitions; window width of 10; profiling sample of 4000)*

and attack samples. We have also shown that proportional power models may also be recovered under the same assumptions, leading to successful correlation DPA attacks which are generally more efficient and almost as robust as the 'partition-based' strategies. Neither are suitable for the task of *evaluation*, which requires considering 'worst case' attacks in ideal scenarios, but they do provide further insight into the capabilities of attackers with limited powers.

Avenues for further work include exploring whether other clustering algorithms are able to improve on the observed example results, and whether there exist attack strategies better able to deal with the particular distortion of misalignment within an acquisition.

6 Acknowledgements

The authors would like to thank Thomas Korak, Thomas Plos and Michael Hutter at TU Graz for supplying us with data from the TAMPRES project [1,13]. The authors have been supported by an EPSRC Leadership Fellowship EP/I005226/1.

References

1. TAMPRES: Tamper Resistant Sensor Nodes. `http://www.tampres.eu`, 2009–2013.
2. C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Template Attacks in Principal Subspaces. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006.
3. K. Baddam and M. Zwolinski. Evaluation of Dynamic Voltage and Frequency Scaling as a Differential Power Analysis Countermeasure. In *20th International Conference on VLSI Design*, pages 854–862. IEEE Computer Society, 2007.
4. L. Batina, B. Gierlichs, and K. Lemke-Rust. Differential Cluster Analysis. In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 112–127. Springer, 2009.
5. L. Batina, J. Hogenboom, and J. van Woudenberg. Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In O. Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 383–397. Springer Berlin / Heidelberg, 2012.
6. S. Chari, J. Rao, and P. Rohatgi. Template Attacks. In B. Kaliski, Ç. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 51–62. Springer Berlin / Heidelberg, 2003.
7. O. Choudhary and M. Kuhn. Template Attacks on Different Devices. In *COSADE 2014 (to appear)*, volume 8622 of *LNCS*. Springer Berlin Heidelberg, 2014.
8. M. Elaabid and S. Guilley. Portability of templates. *J. Cryptographic Engineering*, 2(1):63–74, 2012.

9. D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 7–18, 2003.
10. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis: A Generic Side-Channel Distinguisher. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer–Verlag Berlin, 2008.
11. A. Heuser and M. Zohner. Intelligent Machine Homicide. In W. Schindler and S. Huss, editors, *COSADE 2012*, volume 7275 of *LNCS*, pages 249–264. Springer Berlin Heidelberg, 2012.
12. G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptographic Engineering*, 1(4):293–302, 2011.
13. T. Korak, T. Plos, and M. Hutter. Attacking an AES-Enabled NFC Tag: Implications from Design to a Real-World Scenario. In W. Schindler and S. A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design*, volume 7275 of *LNCS*, pages 17–32. Springer Berlin Heidelberg, 2012.
14. L. Lerman, G. Bontempi, and O. Markowitch. Power analysis attack: an approach based on machine learning. *IJACT*, 3(2):97–115, 2014.
15. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
16. S. Mangard, E. Oswald, and F.-X. Standaert. One for All – All for One: Unifying Standard DPA Attacks. *IET Information Security*, 5(2):100–110, 2011.
17. C. Rechberger and E. Oswald. Practical Template Attacks. In C. H. Lim and M. Yung, editors, *WISA 2004*, volume 3325 of *LNCS*, pages 440–456. Springer, 2004.
18. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, D. Kamel, and D. Flandre. A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 109–128. Springer, 2011.
19. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. Rao and B. Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer Berlin / Heidelberg, 2005.
20. F.-X. Standaert, B. Gierlichs, and I. Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In P. Lee and J. Cheon, editors, *ICISC 2008*, volume 5461 of *LNCS*, pages 253–267. Springer Berlin / Heidelberg, 2009.
21. F.-X. Standaert, T. G. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In A. Joux, editor, *EUROCRYPT '09*, volume 5479 of *LNCS*, pages 443–461, Berlin, Heidelberg, 2009. Springer–Verlag.
22. J. G. van Woudenberg, M. F. Witteman, and B. Bakker. Improving Differential Power Analysis by Elastic Alignment. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *LNCS*, pages 104–119. Springer Berlin Heidelberg, 2011.
23. 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 Berlin / Heidelberg, 2009.
24. C. Whitnall and E. Oswald. A Fair Evaluation Framework for Comparing Side-Channel Distinguishers. *J. Cryptographic Engineering*, 1(2):145–160, August 2011.
25. C. Whitnall, E. Oswald, and F.-X. Standaert. The Myth of Generic DPA...and the Magic of Learning. In J. Benaloh, editor, *CT-RSA*, volume 8366 of *LNCS*, pages 183–205. Springer, 2014.
26. D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.
27. S. Yang, W. Wolf, N. Vijaykrishnan, D. Serpanos, and Y. Xie. Power attack resistant cryptosystem design: a dynamic voltage and frequency switching approach. In *Design, Automation and Test in Europe, 2005. Proceedings*, volume 3], doi=10.1109/DATE.2005.241, ISSN=1530-1591, pages 64–69, March 2005.