Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits

Marios O Choudary and Markus G. Kuhn

Computer Laboratory, University of Cambridge
{osc22,mgk25}@cl.cam.ac.uk

Abstract. Template attacks and stochastic models are among the most powerful side-channel attacks. However, they can be computationally expensive when processing a large number of samples. Various compression techniques have been used very successfully to reduce the data dimensionality prior to applying template attacks, most notably Principal Component Analysis (PCA) and Fisher's Linear Discriminant Analysis (LDA). These make the attacks more efficient computationally and help the profiling phase to converge faster. We show how these ideas can also be applied to implement stochastic models more efficiently, and we also show that they can be applied and evaluated even for more than eight unknown data bits at once.

Keywords: side-channel attacks · template attack · stochastic model · PCA · LDA

1 Introduction

The most powerful side-channel attacks for inferring secret data (passwords, cryptographic keys, etc.) processed inside tamper-resistant hardware use profiling. An attacker first characterizes the signals leaking out of a device while it processes known data values, thereby measuring their probabilistic relationship with the resulting unintended power-supply or electromagnetic emissions (*profiling phase*). The attacker can then use this leakage model to determine the maximum-likelihood data values from the signals leaking out of an identical device that processes unknown data (*attack phase*).

Two such profiling techniques have been described in the literature: the "template attack" [1] and the "stochastic model" [2]. Template attacks are very general in that they use all available information from the side-channel traces to form a probabilistic model for each possible data value (Section 2.1). In contrast, the stochastic method models the leakage through a small number of functions of a data word (e.g. the value of each bit), resulting in fewer parameters to estimate, thereby trading generality of the model for efficiency of profiling (Section 2.2).

One of the main difficulties with implementing these attacks is dealing with a large number of leakage variables, such as oscilloscope traces with thousands of samples. Several compression techniques have been proposed to reduce the

dimensionality of leakage traces, while preserving most of the side-channel information (Section 3). Particularly successful were the application of Principal Component Analysis (PCA) [3] and Fisher's Linear Discriminant Analysis (LDA) [7] to this end. Last year [12], we presented very efficient ways of implementing template attacks with both, and demonstrated in a detailed comparison that those are the most efficient techniques proposed so far.

The question arises, whether similar benefits can be achieved with the stochastic methods [7]. In this paper, we show how to do so, in particular how to adapt the PCA and LDA methods to stochastic models (Section 4). We propose four efficient ways for estimating the PCA and LDA parameters, to preserve the overall efficiency of the stochastic method.

We then use the *Grizzly* dataset [12], which provides real data from an unprotected 8-bit microcontroller, to evaluate all our methods for implementing stochastic models, and we also compare them with template attacks (Section 5). The results show that our methods provide indeed very efficient implementations of stochastic models, while preserving their profiling efficiency.

Finally, we demonstrate how to profile and evaluate stochastic models simultaneously for more than eight bits (Section 6), and we show that our applications of LDA and PCA are particularly helpful in this context.

2 Profiled Attacks

In a profiled attack (template or stochastic model), we need physical access to a pair of identical devices, which we refer to as the *profiling* and the *attacked* device. We wish to infer some secret value $k\star \in \mathcal{S}$, processed by the attacked device at some point. For an 8-bit microcontroller, $\mathcal{S} = \{0, \ldots, 255\}$ might be the set of possible byte values manipulated by a particular machine instruction.

We assume that we determined the approximate moments of time when the secret value $k\star$ is manipulated and we are able to record signal traces (e.g. supply current or electro-magnetic waveforms) around these moments. We refer to these traces as "raw" *leakage vectors*, which we write as $\mathbf{x}^{\mathrm{r}\prime} = [x_1, \ldots, x_{m^{\mathrm{r}}}]$, where $x_j \in \mathbb{R}$ $(1 \le j \le m^{\mathrm{r}})$ is a sample at time index j.[1]

2.1 Template Attacks

During the *profiling* phase we record n_{p} leakage vectors $\mathbf{x}^{\mathrm{r}}_{ki} \in \mathbb{R}^{m^{\mathrm{r}}}$ $(1 \le i \le n_{\mathrm{p}})$ from the profiling device for each possible value $k \in \mathcal{S}$, and combine these as row vectors $\mathbf{x}^{\mathrm{r}}_{ki}{}'$ in the leakage matrix $\mathbf{X}^{\mathrm{r}}_k \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}}$.

Typically, the raw leakage vectors $\mathbf{x}^{\mathrm{r}}_{ki}$ provided by the data acquisition device contain a large number m^{r} of samples (random variables), due to high sampling rates used. Therefore, we might *compress* them before further processing, as explained in Section 3. We refer to such compressed leakage vectors as $\mathbf{x}_{ki} \in \mathbb{R}^m$ and combine all of these as rows into the compressed leakage matrix $\mathbf{X}_k \in \mathbb{R}^{n_{\mathrm{p}} \times m}$. (Without a compression step, we would have $\mathbf{X}_k = \mathbf{X}^{\mathrm{r}}_k$ and $m = m^{\mathrm{r}}$.)

[1] Throughout this paper $\mathbf{x}'$ is the transpose of $\mathbf{x}$.

Using $\mathbf{X}_k$ we can compute the template parameters $\bar{\mathbf{x}}_k \in \mathbb{R}^m$ and $\mathbf{S}_k \in \mathbb{R}^{m \times m}$ for each possible value $k \in \mathcal{S}$ as

$$\bar{\mathbf{x}}_k = \tfrac{1}{n_\mathrm{p}} \sum_{i=1}^{n_\mathrm{p}} \mathbf{x}_{ki}, \quad \mathbf{S}_k = \tfrac{1}{n_\mathrm{p}-1} \sum_{i=1}^{n_\mathrm{p}} (\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)' = \frac{1}{n_\mathrm{p}-1} \widetilde{\mathbf{X}_k}' \widetilde{\mathbf{X}_k}, \quad (1)$$

$\bar{\mathbf{x}}_k$ is the sample mean, $\mathbf{S}_k$ is the sample covariance matrix and $\widetilde{\mathbf{X}_k}$ is the leakage matrix $\mathbf{X}_k$ with $\bar{\mathbf{x}}_k$ subtracted from each row.

In the *attack* phase, we try to infer the secret value $k\star \in \mathcal{S}$ processed by the attacked device. We obtain n_a leakage vectors $\mathbf{x}_i \in \mathbb{R}^m$ from the attacked device, using the same recording technique and compression method as in the profiling phase, resulting in the leakage matrix $\mathbf{X}_{k\star} \in \mathbb{R}^{n_\mathrm{a} \times m}$. Then, for each $k \in \mathcal{S}$, we compute a *discriminant score* $\mathrm{d}(k \mid \mathbf{X}_{k\star})$, and try all $k \in \mathcal{S}$ on the attacked device, in order of decreasing score (optimised brute-force search, e.g. for a password or cryptographic key), until we find the correct $k\star$. If the leakage vectors $\mathbf{x}_i$ can be modeled well by a multivariate normal distribution, which is generally the case and what we also observed in our experiments, then the classic approach is to use a discriminant based on the probability density function (pdf) of this distribution:

$$\mathrm{d}^{\mathrm{joint}}_{\mathrm{PDF}}(k \mid \mathbf{X}_{k\star}) = \prod_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \frac{1}{\sqrt{(2\pi)^m |\mathbf{S}_k|}} \exp\left(-\frac{1}{2} (\mathbf{x}_i - \bar{\mathbf{x}}_k)' \mathbf{S}_k^{-1} (\mathbf{x}_i - \bar{\mathbf{x}}_k) \right). \quad (2)$$

However, if the actual covariance $\mathbf{\Sigma}$ is independent of k, we can use a *pooled* sample covariance matrix [10,12]

$$\mathbf{S}_{\mathrm{pooled}} = \frac{1}{|\mathcal{S}|(n_\mathrm{p}-1)} \sum_{k \in \mathcal{S}} \sum_{i=1}^{n_\mathrm{p}} (\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)(\mathbf{x}_{ki} - \bar{\mathbf{x}}_k)', \quad (3)$$

to better estimate $\mathbf{\Sigma}$, and then use the discriminant score [6,12]

$$\mathrm{d}^{\mathrm{joint}}_{\mathrm{LINEAR}}(k \mid \mathbf{X}_{k\star}) = \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\mathrm{pooled}} \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) - \frac{n_\mathrm{a}}{2} \bar{\mathbf{x}}_k' \mathbf{S}^{-1}_{\mathrm{pooled}} \bar{\mathbf{x}}_k, \quad (4)$$

which avoids numerical pitfalls and is very efficient, being linear in $\mathbf{x}_i$. Throughout our experiments, described in the following sections, we observed that the covariances $\mathbf{S}_k$ are indeed similar. Particular implementations that cause the covariances $\mathbf{S}_k$ to be significantly different are outside the scope of this paper.

2.2 Stochastic Models

Stochastic models were introduced by Schindler et al. [2] as another kind of profiled attack, where the profiling phase can be more efficient than for template attacks. Here, we assume that each sample x_j $(1 \le j \le m)$ of a leakage trace $\mathbf{x}_i$

is modeled as a combination of a deterministic part $\delta_j(k)$, which takes as input a value k, and a random part ρ_j, which models the noise:[2]

$$x_j = \delta_j(k) + \rho_j. \tag{5}$$

This model can be used to attack any target k, similarly to the template attacks in the previous section.

The deterministic function $\delta_j(k)$ is modeled as a linear combination of base functions $g_{jb} : \mathcal{S} \to \mathbb{R}$, with

$$\delta_j(k) = \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k), \tag{6}$$

where $\beta_{jb} \in \mathbb{R}$. The essential idea behind stochastic models is to find a good set of base functions that matches well the leakage of the values k. A common and generally good option for 8-bit architectures is to use the set of $u = 9$ base functions known as $\mathcal{F}_9$, for which $g_{j0}(k) = 1$ and $g_{jb}(k) = \mathrm{bit}_b(k)$. We used $\mathcal{F}_9$ successfully in our 8-bit experiments (Section 5), but in some cases, including XORs between bits [2,4], can improve results (Section 6).

During profiling, instead of acquiring n_p leakage traces $\mathbf{x}^\mathrm{r}_{ki}$ for each candidate k and then use (1,3) to compute the mean vectors $\bar{\mathbf{x}}_k$ and covariance $\mathbf{S}_\mathrm{pooled}$ needed for template attacks, we only use a *total* of N leakage traces $\mathbf{x}^\mathrm{r}_i \in \mathbb{R}^{m^\mathrm{r}}$ from a uniform distribution of the values $k \in \mathcal{S}$. As with template attacks, we generally compress these leakage traces to obtain the compressed traces $\mathbf{x}_i \in \mathbb{R}^m$ ($m \ll m^\mathrm{r}$, see Section 4). Then, we combine all these leakage traces into the leakage matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$ and let k^i represent the value of k corresponding to the trace $\mathbf{x}_i$. Next, for each sample index $j \in \{1, \ldots, m\}$ we build the matrix

$$\mathbf{F}_j = \begin{bmatrix} g_{j0}(k^1) & g_{j1}(k^1) & \ldots & g_{ju-1}(k^1) \\ g_{j0}(k^2) & g_{j1}(k^2) & \ldots & g_{ju-1}(k^2) \\ \vdots & \vdots & \ddots & \vdots \\ g_{j0}(k^N) & g_{j1}(k^N) & \ldots & g_{ju-1}(k^N) \end{bmatrix} \tag{7}$$

and use the stochastic model

$$\mathrm{col}_j(\mathbf{X}) = \mathbf{d}_j + \mathbf{r}_j = \mathbf{F}_j \mathbf{v}_j + \mathbf{r}_j, \tag{8}$$

[2] The original description [2] used a deterministic function $\delta_j(d_i, k)$ with two parameters, to capture any combination of a plaintext value d_i and key value k in an encryption algorithm, and then used a mapping function that reduced this combination into a value to be modeled by the set of base functions g_{jb}. However, the most common mapping is the XOR between d_i and k [2,8] or the XOR between these and a mask value [5]. Therefore, in most cases, a single value (e.g. the XOR result) is modeled by the base functions. If we want to target several values (e.g. for masking [2,5] one might use base functions that depend on both a mask y and the XOR between this mask, a plaintext and a key), we simply concatenate the bits of these values (e.g. $k = [\text{bits mask}|\text{bits XOR}]$).

where $\mathrm{col}_j(\mathbf{X})$ contains the leakage samples x_j of all traces $\mathbf{x}_i \in \mathbf{X}$, $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$, $\mathbf{r}_j{}' = [\rho_j^1, \ldots, \rho_j^N]$, and $\mathbf{d}_j{}' = [\delta_j(k^1), \ldots, \delta_j(k^N)]$. To find the vector of coefficients $\mathbf{v}_j$, we try to minimize the distance $\|\mathrm{col}_j(\mathbf{X}) - \mathbf{F}_j\mathbf{v}_j\|^2$, leading to the solution

$$\mathbf{v}_j = (\mathbf{F}_j{}'\mathbf{F}_j)^{-1}\mathbf{F}_j{}'\mathrm{col}_j(\mathbf{X}). \tag{9}$$

Note that the matrix inversion in (9) requires $\mathrm{rank}(\mathbf{F}_j) = u$ [12], that is $\mathbf{F}_j$ must have u independent rows and columns.

In practice, we may use the same set of base functions (e.g. $\mathcal{F}_9$) for all samples j (or at least for a subset of all samples). In this case, we can drop the subscript j from (7) and use the same $\mathbf{F}$ for all samples j, turning (8) into $\mathbf{X} = \mathbf{F}\mathbf{V} + \mathbf{R}$, allowing us to compute all the coefficients at once as

$$\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m] = (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}, \tag{10}$$

which is computationally more efficient. The coefficient vectors $\mathbf{v}_j$, computed with either (9) or (10), can be used with (6) to compute the deterministic part $\delta_j(k)$ of a sample x_j for any value k. Note that this deterministic part is assumed to be noise-free, since the noise is captured by the term ρ_j. Therefore, as mentioned also by Gierlichs et al. [4], we can use the values $\delta_j(k)$ to compute the stochastic mean vectors $\hat{\mathbf{x}}_k \in \mathbb{R}^m$ as

$$\hat{\mathbf{x}}_k' = [\delta_1(k), \ldots, \delta_m(k)]. \tag{11}$$

While these correspond to the template mean vectors $\bar{\mathbf{x}}_k$ from (1), they depend very much on the choice of base functions.

In order to also use the noise information, we need to compute a covariance matrix $\hat{\mathbf{S}} \in \mathbb{R}^{m \times m}$, similar to the pooled covariance $\mathbf{S}_{\mathrm{pooled}}$ from (3). The available N traces that were used to estimate the coefficients $\mathbf{v}_j$ are good for this purpose[3], since in (5) the deterministic part $\delta_j(k)$ approximates the noise-free part, *common* to all the N traces Therefore, the noise vector $\mathbf{z} \in \mathbb{R}^m$ *specific* to each trace $\mathbf{x}_i$ can be computed as

$$\mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_m^i],\ \rho_j^i = x_j^i - \delta_j(k^i). \tag{12}$$

These vectors can then be used to compute the noise matrix

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1{}' \\ \vdots \\ \mathbf{z}_N{}' \end{bmatrix} = \begin{bmatrix} \rho_1^1 & \ldots & \rho_m^1 \\ \vdots & \ddots & \vdots \\ \rho_1^N & \ldots & \rho_m^N \end{bmatrix}, \tag{13}$$

[3] Schindler et al. [2], as well as following publications [5,8], suggest to use an *additional* disjoint training set of N_2 traces to compute the covariance matrix $\hat{\mathbf{S}}$. However, this requirement was never clearly motivated. In the appendix of an extended version of this paper [14] we show results that sustain our claim.

and finally, we can estimate the covariance matrix as

$$\hat{\mathbf{S}} = \frac{1}{N-1}\sum_{i=1}^{N}\mathbf{z}_i{\mathbf{z}_i}' = \frac{1}{N-1}\mathbf{Z}'\mathbf{Z}. \tag{14}$$

In the attack step, we proceed as in template attacks, using the linear discriminant from (4), but replacing the template mean vectors $\bar{\mathbf{x}}_k$ with the vectors $\hat{\mathbf{x}}_k$ from (11), and the pooled covariance $\mathbf{S}_{\text{pooled}}$ with the covariance $\hat{\mathbf{S}}$ from (14).

3 Compression Methods for Template Attacks

As mentioned earlier, during a profiled attack we should first compress the leakage traces $\mathbf{x}_i^{\text{r}} \in \mathbb{R}^{m^{\text{r}}}$ into $\mathbf{x}_i \in \mathbb{R}^m$ ($m \ll m^{\text{r}}$), in order to reduce the number of variables involved while at the same time keeping as much information as possible. It turns out that the choice of compression method is an essential step for the success of profiled attacks. The first proposed methods [1] relied on selecting some samples that maximise the data-dependent signal, but this can be error-prone. Later, Principal Component Analysis (PCA) [3] and Fisher's Linear Discriminant Analysis (LDA) [7] helped to maximise the information used in the attack step with a very small number m of samples. Last year [12], we provided a detailed analysis of these compression methods in the context of template attacks, and showed that LDA can provide a significantly better attack than the sample selection methods. Below we briefly describe these methods in the context of template attacks, and in Section 4 we show how to adapt them efficiently for use with stochastic models.

3.1 Sample Selection

For the sample selection method we first compute a signal strength estimate s_j for each sample j ($1 \le j \le m^{\text{r}}$), and then select some of the samples having the largest s_j. We used either one sample per clock (*1ppc*) or 20 samples per clock (*20ppc*) among the 5% samples having the highest s_j. Common estimates s_j are the difference of means (DOM) [1] (which can also be computed using the absolute difference [12], as we do in this paper), the Signal to Noise Ratio (SNR) [9] and the sum of squared pairwise t-differences (SOST) [4].

3.2 PCA

For PCA, we define the sample *between groups* matrix

$$\mathbf{B} = \sum_{k\in\mathcal{S}}(\bar{\mathbf{x}}_k^{\text{r}} - \bar{\mathbf{x}}^{\text{r}})(\bar{\mathbf{x}}_k^{\text{r}} - \bar{\mathbf{x}}^{\text{r}})' \in \mathbb{R}^{m^{\text{r}}\times m^{\text{r}}}, \tag{15}$$

where $\bar{\mathbf{x}}_k^{\text{r}} = \frac{1}{n_{\text{p}}}\sum_{i=1}^{n_{\text{p}}}\mathbf{x}_{ki}^{\text{r}}$ are the mean vectors over the *raw* traces $\mathbf{x}_{ki}^{\text{r}}$ and $\bar{\mathbf{x}}^{\text{r}} = \frac{1}{|\mathcal{S}|}\sum_{k\in\mathcal{S}}\bar{\mathbf{x}}_k^{\text{r}}$. Then, we obtain the first m eigenvectors $[\mathbf{u}_1, \ldots, \mathbf{u}_m] =$

$\mathbf{U}^m \in \mathbb{R}^{m^r \times m}$ of $\mathbf{B}$, which contain most of the information about the means, i.e. that can be used to separate well the mean vectors $\bar{\mathbf{x}}_k^r$. For this purpose, we can use the Singular Value Decomposition (SVD) $\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{U}'$, where $\mathbf{D}$ is a diagonal matrix having the eigenvalues (corresponding to $\mathbf{U}$) on its diagonal, and retain only the first m columns of $\mathbf{U}$.[4] We can use visual inspection of the eigenvalues [3], the cumulative percentage of variance [12], or we can also consider the DC contribution of each of the eigenvectors $\mathbf{u}_j$ [13], to select the best value m. Finally, we can compute the projected leakage matrix

$$\mathbf{X}_k = \mathbf{X}_k^r \mathbf{U}^m \tag{16}$$

and obtain the PCA-based template parameters ($\bar{\mathbf{x}}_k$, $\mathbf{S}_{\text{pooled}}$) using (1,3).

3.3 LDA

For LDA, we use the between groups matrix $\mathbf{B}$ and the pooled covariance $\mathbf{S}_{\text{pooled}}$ from (3), computed from the *raw* traces $\mathbf{x}_i^r$, and combine the eigenvectors $\mathbf{a}_j \in \mathbb{R}^{m^r}$ of $\mathbf{S}_{\text{pooled}}^{-1}\mathbf{B}$ into the matrix $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_m]$. Then, we use the diagonal matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$, with $Q_{jj} = (\frac{1}{\mathbf{a}_j{}' \mathbf{S}_{\text{pooled}} \mathbf{a}_j})^{\frac{1}{2}}$, to scale the matrix of eigenvectors $\mathbf{A}$ and obtain $\mathbf{U}^m = \mathbf{A}\mathbf{Q}$. Finally, we use $\mathbf{U}^m$ to project the raw leakage matrices as $\mathbf{X}_k = \mathbf{X}_k^r \mathbf{U}^m$. Using this approach, the resulting covariance matrix of the projected traces becomes the identity matrix, so we only need to use the template mean vectors $\bar{\mathbf{x}}_k$ obtained from (1).

4 Compression Methods for Stochastic Models

4.1 Sample Selection

All the sample selection methods from Section 3 can be adapted for stochastic models by using (11) and (14) to compute the stochastic mean vectors $\hat{\mathbf{x}}_k$ and covariance matrix $\hat{\mathbf{S}}$, and then using these to obtain the desired signal-strength estimate s_j. In addition, Schindler et al. [2] proposed to use $s_j = \sum_{b=1}^{u-1} \beta_{jb}^2$, i.e. the norm of the data-dependent coefficients, which we refer to as *bnorm* in this paper. We used this sample selection method with stochastic models.

4.2 PCA and LDA

Using PCA, and in particular LDA, significantly improved the application of template attacks, and Standaert et al. [7] mentioned that *"Combining data dimensionality reduction techniques with stochastic models is a scope for further research."* However, until now, the sole published attempt to apply PCA to stochastic models, by Heuser et al. [11], is inefficient. As we have shown earlier,

[4] Archambeau et al. [3] show an alternative method for obtaining the matrix $\mathbf{U}$, that can be more efficient when $m^r > |\mathcal{S}|$. This is generally the case, when attacking an 8-bit target, but may not hold when k is a 16-bit target, as in Section 6.

for template attacks, the goal of PCA is to find the eigenvectors $\mathbf{u}_j$ such that the projection in (16) maximises the distance between the compressed traces corresponding to different values k. Instead of using the eigenvectors of $\mathbf{B}$ ("supervised approach"), Heuser et al. [11] used those of the *raw* covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}}$, computed as in (14), to project the leakage traces. While this removes the correlation between leakage samples, it does not maximise the discrimination between means, since the matrix $\hat{\mathbf{S}}^{\mathrm{r}}$ contains no information about the different *raw* mean vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$, obtained from (11), thereby forming an "unsupervised approach". We suspect that the lack of 'mean' information in $\hat{\mathbf{S}}^{\mathrm{r}}$ is also the reason why only the first eigenvalue was significant in the results of Heuser et al., which lead them to use a univariate attack. We verified that, for the *Grizzly* dataset [12], this *unsupervised PCA* method provides no useful attack (i.e. the guessing entropy did not decrease).

We now provide four efficient methods for implementing PCA and LDA with stochastic models. All these methods work in three main steps. In the first step, for which we offer two methods (labelled "S" and "T" below), we compute the matrix $\hat{\mathbf{B}}$, as an approximation of the *between groups* matrix $\mathbf{B}$ from (15), and the *raw* covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}}$ (only needed for LDA). Next, we use either PCA or LDA to obtain the matrix of eigenvectors $\mathbf{U}^m$, and use that to compress the *raw* leakage matrix $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$ into $\mathbf{X} \in \mathbb{R}^{N \times m}$. Finally, for the third step, we use the stochastic model, on the compressed (projected) traces, to model each sample x_j of a *compressed* trace $\mathbf{x}_i = [x_1, \ldots, x_m] \in \mathbf{X}$.

Note that the "S" methods apply the stochastic method *twice*, once on raw traces and once on compressed traces, placing the PCA or LDA compression algorithm into a *stochastic model sandwich*. The general method is shown in Figure 1, Algorithm A.

S-PCA Our first PCA method for stochastic models (named *S-PCA*) relies on the stochastic model from Section 2.2, to build the mean vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$ of the *raw* traces. In the first step, we use these vectors to compute $\hat{\mathbf{B}}$ (see Algorithm B), and in the second step, we obtain $\mathbf{U}^m$ as the eigenvectors of $\hat{\mathbf{B}}$ (see Section 3.2).

T-PCA Our second PCA method for stochastic models (*T-PCA*) is based on the observation that the matrix $\mathbf{B}$ in (15) may be approximated from a subset $\mathcal{S}_s \subset \mathcal{S}$ of values k. Therefore, in the first step, we obtain *raw* traces for the subset $\mathcal{S}_s$, and we use the resulting leakage matrices $\mathbf{X}_k^{\mathrm{r}}$ to compute the matrix $\hat{\mathbf{B}}$ (see Algorithm C). In the second step, we obtain $\mathbf{U}^m$ as the eigenvectors of $\hat{\mathbf{B}}$. Note that for this method (as well as for *T-LDA*, described next), we need two sets of *raw* traces: (a) the N traces in $\mathbf{X}^{\mathrm{r}}$ (used in step 2 and then, compressed, in step 3), and (b) the $|\mathcal{S}_s| \cdot n_{\mathrm{p}}$ traces for the matrices $\mathbf{X}_k^{\mathrm{r}}$ ($k \in \mathcal{S}_s$).

S-LDA and T-LDA We also propose two methods for using LDA with stochastic models: *S-LDA* and *T-LDA*. These are very similar to their PCA counterparts, with S-LDA using Algorithm B, and T-LDA using Algorithm C, to com-

Algorithm A: generic for all PCA/LDA methods

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$
Step 1:
Obtain the matrix $\hat{\mathbf{B}}$ (Algorithm B or C), and the matrix $\hat{\mathbf{S}}^{\mathrm{r}}$ (Algorithm D or E, LDA only)
Step 2:
Obtain the matrix $\mathbf{U}^m$ from $\hat{\mathbf{B}}$ (PCA) or $\hat{\mathbf{S}^{\mathrm{r}}}^{-1}\mathbf{B}$ (LDA)
$\mathbf{X} \leftarrow \mathbf{X}^{\mathrm{r}}\mathbf{U}^m$, $\mathbf{X} \in \mathbb{R}^{N \times m}$
Step 3:
Compute $\mathbf{F}$ (same for all samples) ▷ See (7)
$\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}$ where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$
for all $k \in \mathcal{S}$ **do**
  $\hat{\mathbf{x}}'_k = [\delta_1(k), \ldots, \delta_m(k)],\ \delta_j(k) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k)$
end for
for $i \leftarrow 1, \ldots, N$ **do**
  $\mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_m^i],\ \rho_j^i \leftarrow x_j^i - \delta_j(k^i)$
end for
$\mathbf{Z}' = [\mathbf{z}_1, \ldots, \mathbf{z}_N]$
$\hat{\mathbf{S}} \leftarrow \frac{1}{N-1}\mathbf{Z}'\mathbf{Z}$
Use $(\hat{\mathbf{x}}_k, \hat{\mathbf{S}})$ in the attack step

Algorithm B: compute $\hat{\mathbf{B}}$ for S-PCA/S-LDA

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$
Compute $\mathbf{F}$ (same for all samples) ▷ See (7)
$\mathbf{V}^{\mathrm{r}} = [\mathbf{v}_1, \ldots, \mathbf{v}_{m^{\mathrm{r}}}] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}^{\mathrm{r}}$ where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$
for all $k \in \mathcal{S}$ **do**
  $\hat{\mathbf{x}}_k^{\mathrm{r}\prime} = [\delta_1(k), \ldots, \delta_{m^{\mathrm{r}}}(k)],\ \delta_j(k) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k)$
end for
$\hat{\mathbf{x}}^{\mathrm{r}} \leftarrow \frac{1}{|\mathcal{S}|}\sum_{k \in \mathcal{S}} \hat{\mathbf{x}}_k^{\mathrm{r}}$
$\hat{\mathbf{B}} \leftarrow \sum_{k \in \mathcal{S}} (\hat{\mathbf{x}}_k^{\mathrm{r}} - \hat{\mathbf{x}}^{\mathrm{r}})(\hat{\mathbf{x}}_k^{\mathrm{r}} - \hat{\mathbf{x}}^{\mathrm{r}})'$

Algorithm C: compute $\hat{\mathbf{B}}$ for T-PCA/T-LDA

Require: $\mathbf{X}_k^{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}},\ \forall k \in \mathcal{S}_s$
for all $k \in \mathcal{S}_s$ **do**
  $\bar{\mathbf{x}}_k^{\mathrm{r}} \leftarrow \frac{1}{n_{\mathrm{p}}}\sum_{i=1}^{n_{\mathrm{p}}} \mathbf{x}_{ki}^{\mathrm{r}}$
end for
$\bar{\mathbf{x}}^{\mathrm{r}} \leftarrow \frac{1}{|\mathcal{S}_s|}\sum_{k \in \mathcal{S}_s} \bar{\mathbf{x}}_k^{\mathrm{r}}$
$\hat{\mathbf{B}} \leftarrow \sum_{k \in \mathcal{S}_s} (\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}})(\bar{\mathbf{x}}_k^{\mathrm{r}} - \bar{\mathbf{x}}^{\mathrm{r}})'$

Algorithm D: compute $\hat{\mathbf{S}}^{\mathrm{r}}$ for S-LDA

Require: $\mathbf{X}^{\mathrm{r}} \in \mathbb{R}^{N \times m^{\mathrm{r}}}$
Compute $\mathbf{F}$ (same for all samples) ▷ See (7)
$\mathbf{V}^{\mathrm{r}} = [\mathbf{v}_1, \ldots, \mathbf{v}_{m^{\mathrm{r}}}] \leftarrow (\mathbf{F}'\mathbf{F})^{-1}\mathbf{F}'\mathbf{X}^{\mathrm{r}}$ where $\mathbf{v}_j{}' = [\beta_{j0}, \ldots, \beta_{ju-1}]$
for $i \leftarrow 1, \ldots, N$ **do**
  $\mathbf{z}_i{}' = [\rho_1^i, \ldots, \rho_{m^{\mathrm{r}}}^i],\ \rho_j^i \leftarrow x_j^i - \delta_j(k^i)$, $\delta_j(k^i) \leftarrow \sum_{b=0}^{u-1} \beta_{jb} \cdot g_{jb}(k^i)$
end for
$\mathbf{Z}' = [\mathbf{z}_1, \ldots, \mathbf{z}_N]$
$\hat{\mathbf{S}}^{\mathrm{r}} \leftarrow \frac{1}{N-1}\mathbf{Z}'\mathbf{Z}$

Algorithm E: compute $\hat{\mathbf{S}}^{\mathrm{r}}$ for T-LDA

Require: $\mathbf{X}_k^{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{p}} \times m^{\mathrm{r}}},\ \forall k \in \mathcal{S}_s$
for all $k \in \mathcal{S}_s$ **do**
  $\bar{\mathbf{x}}_k^{\mathrm{r}} \leftarrow \frac{1}{n_{\mathrm{p}}}\sum_{i=1}^{n_{\mathrm{p}}} \mathbf{x}_{ki}^{\mathrm{r}}$
end for
$\hat{\mathbf{S}}^{\mathrm{r}} \leftarrow \frac{1}{(n_{\mathrm{p}}-1)|\mathcal{S}_s|}\sum_{k \in \mathcal{S}_s}\sum_{i=1}^{n_{\mathrm{p}}} (\mathbf{x}_{ki}^{\mathrm{r}} - \bar{\mathbf{x}}_k^{\mathrm{r}})(\mathbf{x}_{ki}^{\mathrm{r}} - \bar{\mathbf{x}}_k^{\mathrm{r}})'$

Fig. 1. Algorithms needed to implement PCA and LDA with stochastic models.

pute $\hat{\mathbf{B}}$. The main difference is that, besides the matrix $\hat{\mathbf{B}}$, we also need to compute the covariance matrix $\hat{\mathbf{S}}^{\mathrm{r}} \in \mathbb{R}^{m^{\mathrm{r}} \times m^{\mathrm{r}}}$ of the *raw* traces. Then, we can obtain $\mathbf{U}^m$ from the eigenvectors of $\hat{\mathbf{S}^{\mathrm{r}}}^{-1}\hat{\mathbf{B}}$, as explained in Section 3.3. Algorithms D and E show how to obtain $\hat{\mathbf{S}}^{\mathrm{r}}$ for S-LDA and T-LDA, respectively.

In Figure 2, we show the first four PCA eigenvectors of the *Grizzly* dataset for template PCA, S-PCA, T-PCA with different random subsets $\mathcal{S}_s$, and the unsupervised PCA, along with the coefficients β_{jb}. For the unsupervised PCA, it is clear that the eigenvectors fail to provide useful information. For the other methods, the first two eigenvectors are very similar. This suggests that S-PCA and T-PCA can produce eigenvectors similar to those from template attacks. Note that for S-PCA and S-LDA we can only obtain a maximum of u eigenvectors corresponding to non-zero eigenvalues, because that is the maximum number of independent vectors used in the computation of $\hat{\mathbf{B}}$ (see Algorithm B).

5 Evaluation on 8-bit data

We use the *Grizzly* dataset [12] to compare the template attacks (TA) with stochastic models (SM). The *Grizzly* dataset contains $n_{\mathrm{p}} = 3072$ *raw* traces $\mathbf{x}_{ki}^{\mathrm{r}}$

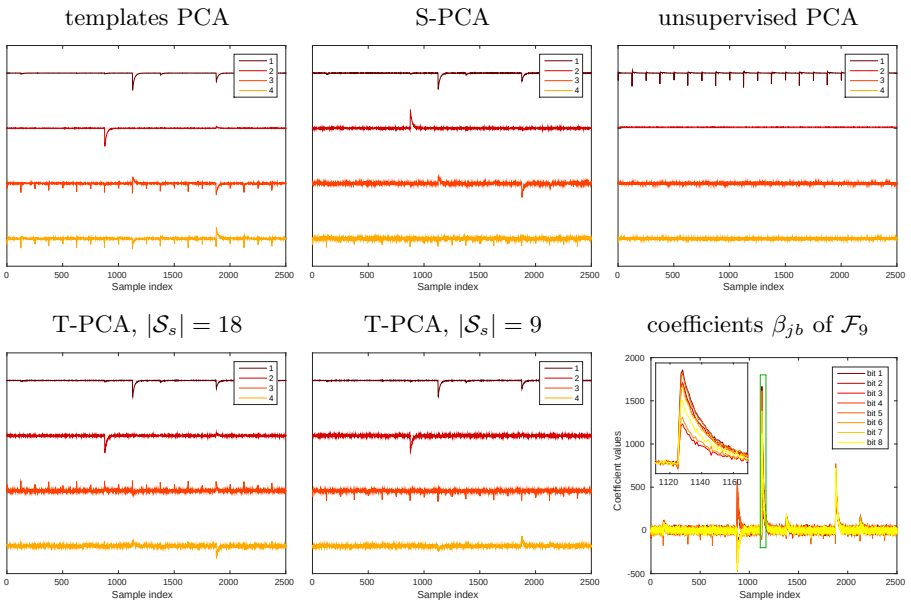

Fig. 2. Normalized eigenvectors for different PCA methods, along with coefficients β_{jb} of $\mathcal{F}_9$ computed via (10).

for each $0 \leq k \leq 255$ (786432 traces in total), which we randomly divide into a training set and an attack set. Each *raw* trace $\mathbf{x}^{\mathrm{r}}_{ki}$ has $m^{\mathrm{r}} = 2500$ samples, corresponding to the current consumption of several consecutive LOAD instructions executed by the unprotected 8-bit Atmel XMEGA 256 A3U microcontroller. A single instruction loads the value k, while the other instructions load the constant value 0. Note that the value k affects the traces over several clock cycles.

5.1 Guessing Entropy

To evaluate the overall *practical* success of a profiled attack we use the *guessing entropy*, following our definition in [12], which estimates the (logarithmic) average cost of an optimised brute-force search. The guessing entropy approximates the expected number of bits of uncertainty remaining about the target value $k\star$, by averaging the results of the attack over all $k\star \in \mathcal{S}$. The lower the guessing entropy, the more successful the attack has been and the less effort remains to search for the correct $k\star$. For all the results shown in this paper, we compute the guessing entropy (g) on 10 random selections of traces $\mathbf{X}_{k\star}$ and plot the average guessing entropy over these 10 iterations.

5.2 Results on 8-bit data

In Figure 3, we show the results of SM using our PCA/LDA methods, along with TA using PCA/LDA for $m = 4$. For TA, we used $n_{\mathrm{p}} = 1000$ traces per

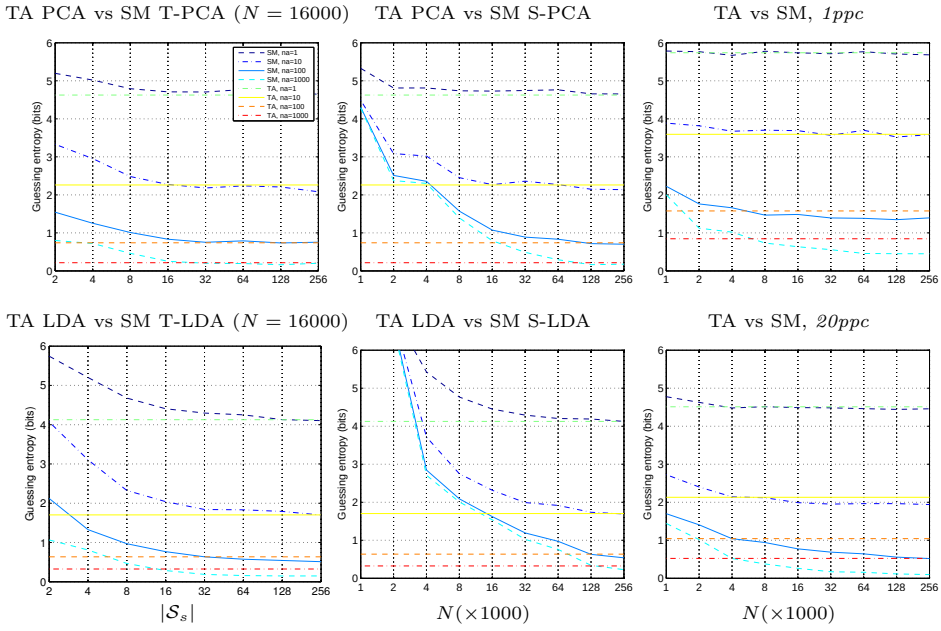

Fig. 3. Comparing TA with SM using PCA, LDA, *1ppc* and *20ppc* with different N and n_a. For TA we used $n_\mathrm{p} = 1000$.

value k during profiling, while for SM we used different N and subsets $\mathcal{S}_s$. We also show the results for SM and TA using *1ppc* ($m = 10$) and *20ppc* ($m = 80$), computed using the absolute difference of means [12].

From these figures, we can observe several things. Firstly, it is clear that all the SM methods provide a guessing entropy equal to or better than their TA equivalent, even when supplied with a much smaller amount of training data. Therefore, our results confirm the observations of Standaert et al. [8], that SM can be at least one order of magnitude more efficient than TA. Theoretically, given enough training data, SM cannot perform better than TA. However, with a limited number of profiling traces, SM may outperform TA when the leakage is modeled well by the chosen base functions. With 256×1000 profiling traces from the *Grizzly* dataset, SM reaches nearly 0-bit guessing entropy with 1000 attack traces, whereas TA does not (Fig. 3, bottom right). Furthermore, if we want to use profiled attacks against data having more than 8 bits, as we show in the next section, the SM may be the only practical choice.

Secondly, we can observe that both S-PCA and T-PCA reach the TA boundary quicker than S-LDA and T-LDA. We believe this to be the case because the PCA methods only depend on $\hat{\mathbf{B}}$ (the approximation of $\mathbf{B}$), while the LDA methods depend on both $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^\mathrm{r}$.

Thirdly, we observe that, for large n_a, the *T-PCA*, *T-LDA*, *S-PCA*, *S-LDA*, and *20ppc* methods provide similar results, but for small n_a, the best results are

obtained by LDA. In particular, note that, using *T-LDA* and *S-LDA*, we can reach 4.1 bits of entropy when $n_a = 1$, while this limit is unreachable for *1ppc* (5.7 bits), *20ppc* (4.5 bits) or PCA (4.7 bits).

From the TA, we knew that PCA and LDA are the most efficient compression methods. Now, we have seen that our PCA/LDA implementations for SM can achieve the same performance. On the other hand, the SM provide more efficient profiling than TA and, moreover, the SM may be the only viable solution to implement profiled attacks against more than 8-bit targets. Therefore, our proposed methods (*S-PCA*, *S-LDA*, *T-PCA* and *T-LDA*) combine the best compression methods (PCA, LDA) with the most efficient profiled attack (SM).

6 Profiled attacks on 16-bit data and more

So far, most publications on profiled attacks have focused on 8-bit attacks. The possibility of attacking 16 bits was mentioned in passing [11], but we are not aware of any public description of the challenges involved in attacking 16-bit data. Therefore, we now consider and demonstrate a profiled 16-bit attack.

6.1 Considerations for the attacker

It is not feasible to mount a template attack on much more than 8 bits, as we need to obtain leakage traces for *each* value k to compute and store the mean vectors $\bar{\mathbf{x}}_k$. However, for the stochastic model, all we need is a selection of traces from a random subset of values k, to estimate the coefficient vectors $\mathbf{v}_j$, from which we can derive any desired stochastic mean vector $\hat{\mathbf{x}}_k$. The remaining limitation is that, in the attack phase, we still need to compute the discriminant $\mathrm{d}_{\mathrm{LINEAR}}$ from (4) over all possible values k. While doing so for $|\mathcal{S}| = 2^{32}$ candidate values is no problem with normal PCs, attempting to do this for 2^{64} candidates would certainly require special hardware.

6.2 Considerations for evaluation laboratories

Even if stochastic methods are practical given a *single* attack trace $\mathbf{x}_i$, a problem that remains, in particular for evaluation labs, is computing the guessing entropy [12], which requires to store n_a traces for *each* value $k\star \in \mathcal{S}$ and run the attack on each of these. This is not practical for values having 16 bits or more. However, one practical solution is to run the attack merely over a subset $\mathcal{S}_s$ of the target values $k\star$ and estimate the expected value of the guessing entropy over these. We refer to this measure as the *sampled guessing entropy* (SGE).

6.3 Efficient attacks and evaluations on more than 8-bit

The complexity of $\mathrm{d}_{\mathrm{LINEAR}}$ is $O(m^2 + n_a \cdot m)$. However, that implies the use of a covariance in (4). But with LDA, we no longer use a covariance matrix (see Section 3), so the complexity of $\mathrm{d}_{\mathrm{LINEAR}}$ reduces to $O(m + n_a \cdot m) =$

$O(n_\mathrm{a} \cdot m)$. Then, an attacker who simply wants to find the most likely k, requires a computation of complexity $O(|\mathcal{S}| \cdot n_\mathrm{a} \cdot m)$ when using LDA (since we need to compute $\mathrm{d}_{\mathrm{LINEAR}}$ for each $k \in \mathcal{S}$), and $O(|\mathcal{S}|(m^2 + n_\mathrm{a} \cdot m))$ when using PCA or sample selection. If n_a is of a lower order than m, then the use of LDA will provide a computational advantage to an attacker. Also, both PCA and LDA will typically work with an m much smaller than that required for sample selection. In our experiments, on the *Grizzly* dataset, we used $m = 4$ for PCA and LDA, while for *20ppc* we use $m = 80$ (sample selection benefits from using many samples per clock [12]). In the extreme case $n_\mathrm{a} = 1$, an attack using LDA will be about 1600 times faster than using *20ppc*, and PCA will be be about 400 times faster than *20ppc*. For larger traces, covering many clock cycles (e.g. for a cryptographic algorithm), we expect this difference to increase. Therefore, our PCA and LDA implementations for SM can offer great computational advantage.[5]

An evaluator who wants to compute the SGE will run the attack for each $k\star \in \mathcal{S}_s$. Therefore, the complexity of the evaluation is $O(|\mathcal{S}_s| \cdot |\mathcal{S}| \cdot n_\mathrm{a} \cdot m)$ for LDA and $O(|\mathcal{S}_s| \cdot |\mathcal{S}| \cdot (m^2 + n_\mathrm{a} \cdot m))$ for PCA or sample selection. However, we can optimise the computation of the SGE by precomputing $\mathbf{y}_k = \hat{\mathbf{x}}_k' \hat{\mathbf{S}}^{-1}$, and $z_k = \mathbf{y}_k{}' \hat{\mathbf{x}}_k$, which require a computation of $O(|\mathcal{S}|m^2)$. With these values, the discriminant $\mathrm{d}_{\mathrm{LINEAR}}$ can be computed as

$$\mathrm{d}^{\mathrm{fast}}_{\mathrm{LINEAR}}(k \mid \mathbf{X}_{k\star}) = \mathbf{y}_k{}' \left(\sum_{\mathbf{x}_i \in \mathbf{X}_{k\star}} \mathbf{x}_i \right) - \frac{n_\mathrm{a}}{2} z_k, \tag{17}$$

which has complexity $O(n_\mathrm{a} \cdot m)$. Therefore, the evaluation of the partial guessing entropy can be done with complexity $O(|\mathcal{S}|m^2 + |\mathcal{S}_s| \cdot |\mathcal{S}| \cdot n_\mathrm{a} \cdot m)$. For PCA, the value m may be comparable to or smaller than $|\mathcal{S}_s|$ and therefore an evaluation using $\mathrm{d}^{\mathrm{fast}}_{\mathrm{LINEAR}}$ will run as fast as an evaluation using LDA. However, if we need to use a sample selection method with very large m, then the evaluation will be considerably slower. Remember also that, while *1ppc* with low m may be as fast as LDA in this case, we confirmed in Figure 3 that both PCA and LDA provide better results than *1ppc*.

These considerations show that the choice of compression method depends also on who will need to use it: an attacker who only wants the correct $k\star$, or an evaluator who wants to know the average attack cost. In both cases, our LDA and PCA methods will help.

6.4 Results on 16-bit data

In order to verify that an attack on 16-bit data is feasible, and to obtain an estimate on the actual run time, we used the same device as in the *Grizzly* dataset: an Atmel XMEGA 256 A3U, with similar data acquisition setup for current traces. In order to obtain 16-bit data, we varied the 8-bit values processed by

[5] We also note that, for SM with sample selection, we should use *bnorm* (see Section 4.1), as that is more computationally efficient than the other methods for estimating the signal-strength estimate s_j.

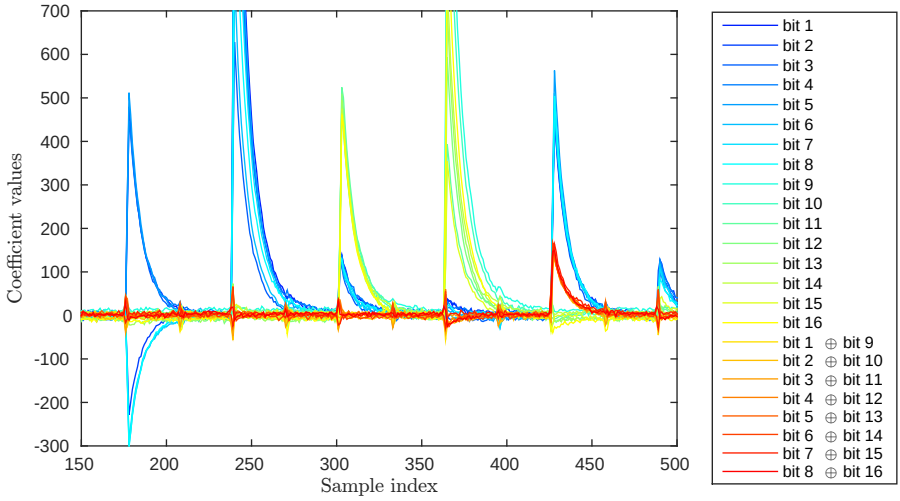

Fig. 4. Contribution of coefficients β_{jb} in $\mathcal{F}_{17}$ (bits 1 to 16) and $\mathcal{F}_{17x}$ ($\mathcal{F}_{17}$ enhanced with XOR between bits of 8-bit halves) for the *Panda* dataset. Pipelining causes leakage of the two 8-bit halves to overlap (e.g. around sample 300). Their consecutive processing also leaks their XOR value (e.g. around sample 430). While the β_{jb} for bits 1–16 are not exactly identical for $\mathcal{F}_{17}$ and $\mathcal{F}_{17x}$, the difference is visually indistinguishable.

two consecutive LOAD instructions, thus obtaining leakage traces that depend on 16-bit values. Using this scenario, we cannot evaluate the limit of SM on a 16-bit parallel bus, but we can evaluate the feasibility of profiled attacks on more than 8 bits of data. For this dataset, which we call *Panda*, we acquired $n_\mathrm{p} = 200$ traces for each of the $2^{16} = 65536$ values $k \in \mathcal{S}$ ($N = 13\,107\,200$ traces in total). Each trace has $m^\mathrm{r} = 500$ samples, recorded with 125 MS/s using the HIRES mode of our Tektronix TDS 7054 oscilloscope (which provides ≈ 10.5 bits per sample by averaging 40 consecutive 8-bit samples acquired internally at 5 GS/s), and contained data over 5 LOAD instructions, of which two contained our target data and the other three processed the constant value 0. We split this data into two sets, for profiling and attack. In addition we also acquired $n_\mathrm{p} = 1000$ traces for a selection of $|\mathcal{S}_s| = 512$ random values k (512000 traces in total), which we used for the estimation of $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^\mathrm{r}$ with T-PCA and T-LDA. For the implementation of the SM we simply extended the set of base functions to include the individual contributions of all the 16 bits of the values k, resulting in the set $\mathcal{F}_{17}$. The contribution of each base function is shown in Fig. 4.

In Figure 5, we show our results for the full 16-bit attack. For our SM versions of PCA and LDA we used $m = 10$. With most methods, the guessing entropy converges after only about $N = 1000 \times 2^4 = 16000$ traces, which confirms the efficiency of stochastic models. S-LDA reduces the guessing entropy below 4 bits when using $n_\mathrm{a} = 100$ traces, which means that, in this case, we can find the correct $k\star$ in a brute-force search attack with at most 16 trials, on average.

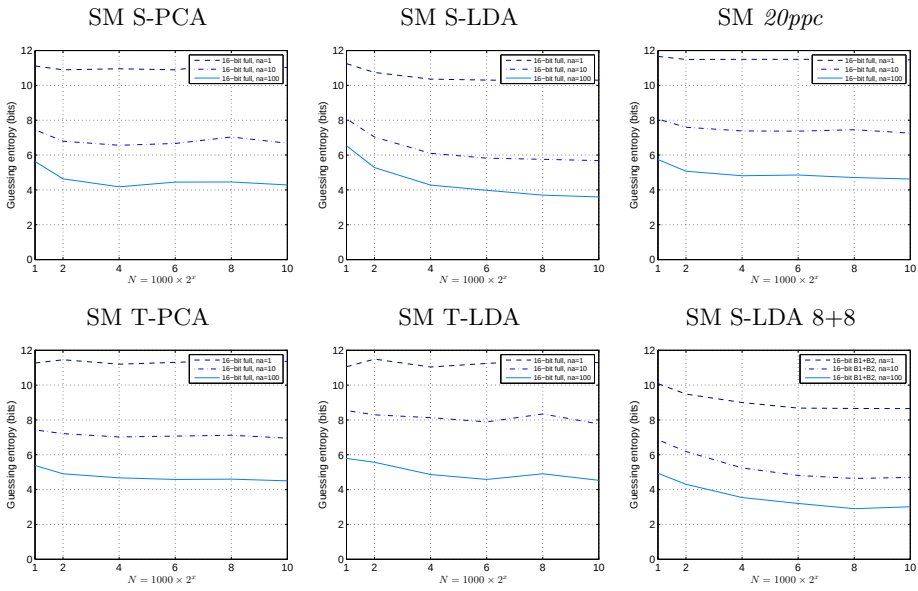

Fig. 5. Results of full 16-bit $\mathcal{F}_{17}$ attack for pipelined data, with *S-PCA*, *S-LDA*, *20ppc*, *T-PCA*, *T-LDA*, and results from *S-LDA* on 8 bits at a time, using different N and n_a. We tried $N = 1000 \cdot 2^x$, where x is the value shown on the logarithmic x-axis. For *T-PCA* and *T-LDA*, we used $|\mathcal{S}_s| = 512$.

S-PCA, S-LDA and T-PCA are better than *20ppc*, but T-LDA is not. Both S-PCA and S-LDA are better than T-PCA and T-LDA, which suggests that the subset of $|\mathcal{S}_s| = 512$ values we used for the estimation of the T-PCA/T-LDA parameters $\hat{\mathbf{B}}$ and $\hat{\mathbf{S}}^{\mathrm{r}}$ was not enough to reach the full potential of PCA and LDA. Therefore, for attacks on more than 8 bits the methods S-PCA and S-LDA may be the best option, as they can use all the available N traces with the stochastic model for both the modeling of the compressed mean vectors $\hat{\mathbf{x}}_k$ (step 3 in Algorithm A), as well as for the modeling of *all* the raw vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$ (lines 3–5 in Algorithm B). This in turn can result in a better estimation of the matrix $\hat{\mathbf{B}}$ (step 1 in Algorithm A), than what can be achieved with a small subset of real vectors $\bar{\mathbf{x}}_k^{\mathrm{r}}$ for the T-PCA and T-LDA methods.

In the bottom-right of Figure 5, we also show the results when performing the SM attack separately, for each of the two bytes of our target value (i.e. during profiling we only consider one byte known, while the varying value of the other represents noise). We computed the results by adding the guessing entropy from each 8-bit attack. This figure shows that, in our particular scenario, performing two 8-bit attacks (each with $\mathcal{F}_9$) provided better results than any of the 16-bit attacks with $\mathcal{F}_{17}$. This could potentially be due to several factors. Firstly, by attacking only 8 bits, there are fewer parameters to be estimated during the attack (e.g. the SM coefficients). Secondly, the signal-to-noise ratio

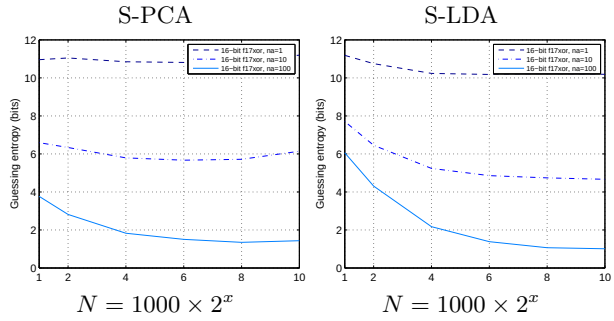

Fig. 6. Results of SM attack using $\mathcal{F}_{17x}$ with S-PCA (left) and S-LDA (right).

in the acquisition setup might have been too low to provide sufficient separation between the $|\mathcal{S}| = 2^{16}$ classes to be distinguished by our classifier. Finally, the base function set $\mathcal{F}_{17}$ may simply not have adequately modeled the leakage. The latter turned out to be the main factor, which was easily fixed. Our 16-bit target value $k = [k_1|k_2]$ is composed of two 8-bit halves (k_1 and k_2), which are processed consecutively in the XMEGA CPU. If these two values pass through the same parts of the circuit, their XOR difference is likely to affect part of the leakage traces. Therefore, we also evaluated an attack where the stochastic model included the XOR between the bits of k_1 and k_2, resulting in the set $\mathcal{F}_{17x}$ (see Fig. 4).

Figure 6 shows the results of our SM attacks using S-PCA (left) and S-LDA (right) with $\mathcal{F}_{17x}$. We see that, using $\mathcal{F}_{17x}$, both S-PCA and S-LDA perform better than with $\mathcal{F}_{17}$. Also, in this case S-LDA reduces the guessing entropy to about one bit, which is far better than any of the other results, including the attack on k_1 and k_2 separately. Therefore, a 16-bit attack can perform better than two 8-bit attacks, if a model is used that also takes into consideration differences between the bits, as we did in $\mathcal{F}_{17x}$.

In Table 1, we show the execution times for the main steps of an evaluation using S-PCA. This table shows that SM attacks are feasible, at least computationally, on 16-bit data. All the steps can be extended for 32-bit data and more. The only steps that depend on the number of bits are the computation of the raw vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$ and the computation of the compressed vectors $\hat{\mathbf{x}}_k$ for all $k \in \mathcal{S}$, and the computation of the SGE. These steps depend *linearly* on k, so a straight-forward extension to 32-bit may require 65536 times more time. That means that, for an attacker who only wants to find what the most likely target $k\star$ is, the attacks would take 24 days for the computation of the raw vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$, 21 days for the computation of the compressed vectors $\hat{\mathbf{x}}_k$ and 15 hours for the attack step. However, it seems that for an evaluator it would be impractical to compute the SGE on 32-bit data for large $|\mathcal{S}_s|$.

Table 1. Approximate time required for the main steps of an evaluation using S-PCA on 16 bits with $\mathcal{F}_{17}$ and $N = 64000$

Step	**time**
Obtaining $\mathbf{V}$ on raw data (Algorithm B, step 2)	40 s
Approximating raw mean vectors $\hat{\mathbf{x}}_k^{\mathrm{r}}$ (Algorithm B, steps 3–5)	32 s
Computing PCA parameters (Algorithm A, step 2)	2 s
Obtaining $\mathbf{V}$ on compressed data (Algorithm A, step 5)	38 s
Obtaining $\hat{\mathbf{x}}_k$ for all k (Algorithm A, steps 6–8)	28 s
Obtaining $\hat{\mathbf{S}}$ (Algorithm A, steps 9–13)	33 s
Compute SGE using $\lvert\mathcal{S}_s\rvert = 256$ with m of the same order as n_{a}	210 s

7 Conclusions

In this paper, we have shown how to implement the PCA and LDA compression methods, which have so successfully boosted the performance of template attacks in the past, also for stochastic models. As both techniques implement coordinate transforms based on singular-value decomposition of covariance matrices, there were two opportunities to apply a stochastic model: first before the compression step, on raw traces, to aid estimating the matrices required by the compression method, and secondly, on the compressed traces that they output, to better estimate the mean vectors for each data value. In addition, we investigated a variant in which the matrices for the compression step are instead estimated directly, as in the template attacks, but using only a subset of all possible data values, which also boosts the performance of the profiling phase.

We have shown that, for 8-bit attacks, our PCA and LDA methods for stochastic models can obtain the same or even better results than their respective implementations on template attacks. Combining the compression efficiency of PCA and LDA with the profiling efficiency of stochastic models allows us to extract the most out of profiled attacks. Moreover, we have shown that, from a computational perspective, LDA can provide a significant advantage to an attacker, which for our experiments may result in an attack step that is 1600 times faster than using sample selection. For an evaluator, both LDA and PCA will be very helpful in obtaining efficient evaluations of profiled attacks.

We also performed an evaluation on 16-bit data, which allowed us to confirm that: (a) our PCA and LDA implementations provide good results, and (b) stochastic attacks are feasible, at least computationally, on 16-bit data. Extrapolating the run-time of our evaluation, even an attack on 32 bits appears computationally feasible (requiring in the order of 45 days for the profiling step and 15 hours for the attack step on our PC).

Our results also showed that two separate 8-bit attacks performed better than a 16-bit attack, which could be attributed to several factors, such as fewer parameters to be estimated, or a limitation in the acquisition setup and attack method to distinguish a 16-bit value. However, when adding the contribution of the XOR between the two 8-bit halves of our target value to the 16-bit model, we obtained better results. This showed that simply expanding the attack to

16 bits is not guaranteed to improve the results, because the larger number of parameters that need to be estimated reduced the accuracy achievable with a given set of traces, and in such situations, an attack targeting two 8-bit halves separately can actually perform better. A 16-bit attack, however, can perform better if a more informative model is used, such as taking into consideration differences between the bits, as we did in $\mathcal{F}_{17x}$, in which case the attack could outperform the individual 8-bit attacks.

Data and Code Availability: In the interest of reproducible research we make available our data and related MATLAB scripts at:

`http://www.cl.cam.ac.uk/research/security/datasets/grizzly/`

Acknowledgement: The first author is a recipient of the Google Europe Fellowship in Mobile Security, and this research was supported in part by this fellowship. The opinions expressed in this paper do not represent the views of Google unless otherwise explicitly stated.

References

1. S. Chari, J. Rao, and P. Rohatgi, "Template Attacks", CHES 2002, Springer, 2003, LNCS 2523, pp 51–62.
2. W. Schindler, K. Lemke, and C. Paar, "A Stochastic Model for Differential Side Channel Cryptanalysis", in CHES 2005, LNCS 3659, pp 30–46.
3. C. Archambeau, E. Peeters, F. Standaert, and J. Quisquater, "Template Attacks in Principal Subspaces", in CHES 2006, Springer, 2006, LNCS 4249, pp 1–14.
4. B. Gierlichs, K. Lemke-Rust, and C. Paar, "Templates vs. Stochastic Methods", in CHES 2006, Springer, LNCS 4249, pp 15–29.
5. K. Lemke-Rust and C. Paar, "Analyzing Side Channel Leakage of Masked Implementations with Stochastic Methods", ESORICS 2007, LNCS 4734, pp 454–468.
6. P. Karsmakers, et al., "Side channel attacks on cryptographic devices as a classification problem", KU Leuven, COSIC, internal report, 2007. `https://www.cosic.esat.kuleuven.be/publications/article-2450.pdf`
7. F.-X. Standaert and C. Archambeau, "Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages", in CHES 2008, Springer, LNCS 5154, pp 411–425.
8. F.-X. Standaert, F. Koeune, and W. Schindler, "How to Compare Profiled Side-Channel Attacks?" ACNS 2009, LNCS 5536, pp 485–498.
9. S. Mangard, E. Oswald, and T. Popp, "Power Analysis Attacks: Revealing the Secrets of Smart Cards", 1st ed., Springer, 2010.
10. D. Oswald and C. Paar, "Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World", in CHES 2011, LNCS 6917, pp 207–222.
11. A. Heuser, M. Kasper, W. Schindler, and M. Stöttinger, "A New Difference Method for Side-Channel Analysis with High-Dimensional Leakage Models" in CT-RSA 2012, LNCS 7178, pp. 365–382.
12. O. Choudary and M. G. Kuhn, "Efficient Template Attacks", CARDIS 2013, Berlin, 27–29 November 2013, LNCS 8419, pp. 253–270.
13. O. Choudary and M. G. Kuhn, "Template Attacks on Different Devices", COSADE 2014, Paris, 14–15 April 2014, LNCS 8622, pp. 179–198.
14. M. O Choudary and M. G. Kuhn, "Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits", extended version, Cryptology ePrint Archive, Report 2014/885, 2014. `https://eprint.iacr.org/2014/885.pdf`.