Time, Memory and Accuracy Tradeoffs in Side-Channel Trace Profiling

Hen Hayoon(✉)and Yossi Oren(✉)

Department of Software and Information Systems Engineering,
Ben-Gurion University, Beersheba, Israel
hayoonh@post.bgu.ac.il and, yos@bgu.ac.iland

Abstract. Template attacks are one of the most powerful classes of side-channel attacks. Template attacks begin with an offline step, in which the side-channel traces are profiled, and decoders are created for each side-channel leak. In this paper, we analyze the compression step of the trace profiling process. This compression step, which is a central part of the decoder's training process, is used to reduce the amount of time, memory consumption, and data required to successfully perform the attack; various practical methods have been proposed for this step, including one which uses an efficient means both for selecting the points of interest (POI) in the power trace and for preprocessing noisy data.

We investigate ways to improve the efficiency of the attack by implementing several compression methods which select the most informative power consumption samples from power traces. We develop a unique dedicated evaluation system to compare the performance of various decoders with different compression methods on real-world power traces. Our findings indicate that our proposed decoder for side-channel traces outperforms the current state of art in terms of speed, resource consumption, and accuracy. We also demonstrate our decoder's effectiveness under resource-constrained conditions, and show that it achieves over 70% accuracy even if there are fewer than 1,000 traces in the profiling phase.

1 Introduction

Side-channel attacks (SCAs) [4,9,15] have been shown to be effective and practical for attacking implementations of cryptographic algorithms. These attacks reveal cryptographic device secrets by observing the physical properties of the device [15]. Adversaries can obtain sensitive information from side-channels, such as the timing of operations, power consumption, electromagnetic emanations, etc. [4,15,17]. When a cryptographic operation is performed, the device emits a data dependent side-channel leak. Leaks are the internal state functions of the device under test (DUT), and they are modulated into a power/EM trace, along with some noise. In constrained devices, such as chip-cards, straightforward implementations of cryptographic algorithms can be broken easily, since the power consumption of the cryptographic device is dependent on the intermediate values of the cryptographic algorithm executed. When the amount of

S. Dolev et al. (Eds.): CSCML 2022, LNCS 13301, pp. 29–46, 2022.
https://doi.org/10.1007/978-3-031-07689-3_3

leakage is smaller relative to the noise, statistical techniques such as DPA are applicable. As stated by Chari et al. [6], DPA relies on the statistical analysis of a large number of samples where the same keying material is used to operate on different data. The most powerful type of side-channel attack is called the template attack (TA). The TA's evaluation of side-channel information relies on a multivariate model of the side-channel traces. This attack contains an offline and online phase, and one of its advantages is that it requires just a few samples in the online phase, and works well even if the DUT's power consumption does not conform to the Hamming weight leakage model.

In order to recover the secret key in a template-based side-channel attack, the attacker's operations consist of three phases. First, in the offline *profiling phase*, a device totally controlled by the attacker and similar to the DUT is profiled and characterized. This DUT analysis, like correlation power analysis (CPA) [4], identifies the position of the leaking operations in the traces by identifying a small section of the power trace T depending only on a few unknown key bits. The profiling phase outputs a series of *decoders*, each mapping a certain set of points of interest (POIs, also known as features) in the trace to a certain set of secret values. The Hamming weight model is an example of the mapped output of this phase [17,25]. The second phase consists of an online *decoding phase*, where the attacker is provided with a few power traces, generally a single one, and uses the decoders created in the profiling phase to recover leak vectors from the power trace. These leak vectors may contain some errors due to noise. The last phase is the *solving phase*, where the correct key is discovered from among the most likely candidates by using the brutre force, the maximum-likelihood method [12] or by the use of a constraint solver [14,25].

1.1 Contribution

Trace compression is the initial step of the profiling phase. In this step the power trace is replaced with a smaller-sized vector, in order to improve the decoder's performance, in terms of the number power traces required, and its overall performance in both the online and offline phases. In this paper, we investigate which parameters for this compression step deliver the best combination of accuracy and runtime performance. Specifically, our paper makes the following contributions: we design and implement a unique evaluation performance system which can analyze the compression step. We then use this system to explore and compare several profiling methods. The profiling methods differ based on the compression techniques and preprocessing model (for the training set traces) utilized. Three well-known and different classical compression methods were implemented. In addition, we implement the method used by the smart decoder proposed in [18] and presented in Sect. 3.4, it offers an efficient searching algorithm to find the most leaking points in the trace using a unique compression method. Finally, we propose our own optimal profiling method, based on the guidelines of [24], as a tradeoff that performs well under conditions of data and resource restrictions.

1.2 Related Work

Weisse et al. [18] presented a method for profiling the training set of power traces into an accurate decoder to be used as part of an algebraic side-channel attack aimed at recovering the secret key. The authors provided two scoring methods used to identify the best leaking points in the trace; their methods are effective when there is limited representation of some values in the training set traces. Kocher et al. [13] were the first to present the difference of means (DOM) as an alternative to correlation for power computation measurements analysis in order to determine the secret keys of a DES operation. Before that, Chari et al. [6] showed the best visualization for the DOM of side-channel samples for a TA with a single sample against an RC4 implementation.

Mangard et al. [15] used the DOM as an alternative to correlation not only for the binary power model but for the Hamming-weight model, directly on power traces to perform a DPA attack on the S-box of an AES implementation [8]. In another chapter of their book the authors suggested the model-based integration of SNR (signal-to-noise ratio) techniques to compress power traces, using the sum of their signal and noise in a defined time interval. Gierlichs et al. [10] suggested the sum of squared pairwise differences (SOSD) instead of the regular sum of pairwise of the DOM for the selection of interesting points in a TA for an SCA against the AES. Rechberger et al. [24] presented a practical TA using an advanced version of the maximum extraction compression method to select the interesting points in a power trace. Instead of choosing the highest points, the authors defined a few properties that must exist at the selected points.

Other authors revealed the importance of feature selection in an SCA and compared them in other scenarios. Zheng et al. [31] compared known feature selection techniques and evaluated their accuracy for profiled SCAs. Picek et al. [23] investigated advanced feature selection techniques from the machine learning domain used to improve attack accuracy, examining the influence of the number of features in the process. Cagli et al. [5] presented an accuracy comparison analysis for feature selection in SCAs through linear, non linear, and neural-network models.

2 Background

In this work we assume a DUT conforming to the Hamming weight power leakage model.

Let k be an encryption key (bytes), and p and c are respectively the plaintext and ciphertext of the cryptographic algorithm, which is uniformly chosen. The multivariate power trace measured is denoted as $\overrightarrow{X} = X_1, ..., X_S$, where S is the number of time samples (also called features).

In the offline phase, the attacker, who controls the DUT, estimates the leakage model using a set of N profiling traces $\overrightarrow{X}_1, ...\overrightarrow{X}_N$ (multi dimensional $S \times N$) and the knowledge of k. In the online phase, additional power traces of the DUT $\overrightarrow{X}_1, ...\overrightarrow{X}_M$ itself are measured (one or more), and the attacker's objective is to recover k from these power traces using signal classification techniques.

2.1 Template Attacks

Template attacks are a method for performing power analysis attacks which works by creating a characterization of a device. The attack usually consists of an offline phase, in which device characterization takes place, followed by a online phase, in which the characterization is used for the attack.

First, a series of templates of all possible operations (i.e., all of the cryptographic algorithms are executed using all of the possible subkey values) is constructed. Then, the attack starts, and the trace of a single operation is captured. Using the templates created, which represent all key values, the side-channel information of the attacked device is classified and assigned to one or more of the templates. The goal is to significantly reduce the number of possible keys, optimally concluding the attack with a single possible value for the secret key.

Building Templates. In the state-of-the-art template attack the power samples are considered dependent by the TA; accordingly, the traces are characterized with the multivariate normal distribution [6,7,15,24,26–28]. The characterization defined by the "template" of the multivariate normal distribution, is the pair (m, CM) where m represents the mean vector and CM represents the covariance matrix for each class. When building templates with the power model, we determine templates for certain sequences of instructions by executing them with different and known data d_i and keys k_j, in order to record the resulting power consumption. Then, we group the corresponding traces to the pair of (d_i, k_j) and estimate (m, CM).

As a result, we obtain a template for every data and key pair. Then, using the template, along with $(m, CM)_{d_i,k_j}$ and the power trace x, we evaluate the probability density function of the multivariate normal distribution :

$$p(x; (m, CM)_{d_i,k_j}) = \frac{exp(-\frac{1}{2} \cdot (x-m)' \cdot CM^{-1} \cdot (x-m))}{\sqrt{(2\pi)^\tau \cdot det(CM)}}$$

The probabilities for each template measure how well they fit to a given trace. If the noise level is sufficiently low, the maximum-likelihood decision rule can be applied, and the template with the highest probability indicates the correct key.

2.2 Dataset

The dataset used in this study is the DPA contest v4 dataset [20], which provides measurements of a masked AES implementation. In that attack contest, the goal was to use the smallest number of power consumption traces to identify the first 128 bits of the encryption key. **The hardware** used for the cipher implementation was the Atmel ATMega-163 smart card, which was sampled using a LeCroy Waverunner 6100A oscilloscope at the rate of 500 MS/s. The dataset provides 100,000 power traces, each of which consists of 435,002 samples and corresponds to the execution of an AES-256 round. **The countermeasure** of the AES-256 implementation was "Rotating S-Box Masking" (RSM) [21]; all

of the power traces provided were running the "AES-RSM" implementation with the same 256-bit key, but different plaintext was used for each measurement. This implementation contains the following features:

- An arbitrary fixed 16-byte mask is added on top of the classic AES. A random offset $0 \leq o \leq 15$ is drawn as the first stage of the encryption process. Let m^0 denote the cyclic rotation of the mask added by offset o.
- The masked plaintext pm is the result of the XOR operation of the 16 bytes of plaintext with m^0. Let pm_i denote the result $pm_i = p_i \oplus m_i^0, 0 \leq i \leq 15$.
- The AddRoundKey phase uses pm for each sub-round key.
- The masked S-boxes, which are derived from the value of m^0 of, are used.
- The ShiftRows and MixColumns sub-round phases are unchanged.

We generated and parsed the plaintext, offsets and key files into comma-separated value (CSV) files, according to the size expected by the parsing code used by [18], and matched them (populating from index 0) with the RSM trace indices, according to the trace index file found on the DPA contest site [21].

2.3 The Hamming Weight Leakage Model

General Assumption. The assumed form of the Hamming weight (HW) leakage of information in power consumption described in [15] is: $P_{total} = P_{exp}(HW(s_i)) + P_n$, where $HW(s_i)$ denotes the HW of the intermediate state byte s_i for a certain leak i, and P_n denotes the noise component which is assumed to be normally distributed with unknown parameters. This form is exactly the probabilistic model used to construct a Bayesian classifier. As was done in [22,24,30], we therefore use a *naive Bayes* (NB) classifier. The NB classifier returns, for each feature, a mean and variance for each class of the 9 possible HW classes (0–8).

AES-RSM Leaks. The leakage model for the AES-RSM implementation is the HW model. The desired leakage of information of the AES-RSM implementation is the Hamming weight of the S-box state bytes they process. The following leaks can be derived from the traces of DPA v4 used in our study: 16 bytes of the masked plaintext pm_i, as describe in Sect. 2.2, 16 bytes of the output of the AddRoundKey computation, 16 bytes of the output of SubBytes, and finally 52 bytes from the MixColumns computation. The first 16 bytes were added by the RSM countermeasure, The rest 84 bytes are the same as enumerated in [18]. In aggregate, there are 100 leaks from 100 intermediate byte values.

3 Compression Methods

The compression step is the first step performed during the profiling phase of an SCA. Compression methods are usually used to reduce the complexity of power analysis attacks, by reducing the length (dimension) of the power traces. This is done in cases in which there are not enough traces for a full rank covariance matrix CM, to cope with computational or memory restrictions, as the size of the CM grows quadratically with the number of samples in the trace.

The motivation for using compression methods stems from the amount of redundancy present in long power traces, as these methods are able to remove this redundancy without significant loss of leak information. To ensure an efficient compression process, it is necessary to know which points in the power traces points the "points of interest" (POIs) and contain information relevant to an attacker. These samples have the highest information leakage, which is reflected in their high correlation to the number of transitions that occur in the chip, where we implicitly assume that the number of transitions depends on the operation performed and on the data being processed. Identification of the POI by the attacker is the first step in device characterization, and this information is used to build the templates.

There are two main compression method approaches: The first is the "selection of samples" approach, which is based on some criteria, and the second is the "usage of linear combinations" of the leakage vectors approach, which based on the principal components or Fisher's linear discriminant.

In this section, we describe methods from both approaches in order to cover a wide range of methods in our performance comparison. We first present a few classical methods; then we discuss the compression method used by the smart decoder proposed in [18], and finally we describe the compression method used in our decoder.

3.1 Principal Component Analysis

Principal component analysis (PCA) is mainly used in multivariate statistics to reduce the dimensionality of a dataset while retaining the most variance [3], by finding patterns within the dataset. PCA searches for linear combinations with the greatest variance, and divides them into principal components (PCs) where the greatest variance is captured by the highest component. The first PC is required to have the greatest variance. The second PC must be orthogonal to the first component while capturing the greatest variance within the dataset in that direction; subsequent components cover less and less of the remaining data variance.

The maximum number of PCs (dimensionality) is equal to the number of samples in the power trace. Choosing the right number of PCs (designated by n) is essential for obtaining optimal results, as shown in [11], by maximizing the variance in the original data and minimizing the reconstruction error of the data transformation.

The PCA method is based on the usage of the linear combinations approach (mentioned above), and we chose to implement it, since according to [7,19], its success, unlike the Linear discriminant analysis (LDA) method, depends on the condition of equal covariance (known as homoscedasticity). We use the MATLAB implementation of PCA [1] which returns a vector containing the percentage of the total variance explained by each PC. Then, the mean of each relevant feature from all of the training traces is calculated.

The training part calculates the principal features by multiplying the coefficient by the difference of each feature with the mean. The vector returned is

the implementation of the "cumulative percentage of total variation" method for choosing n, which is the method recommended by [11,19]; we experiment with n values in the range $[5, 10, 30]$, since the performance measurements with n $<$ 5 and n $>$ 30 are extreme and unstable.

3.2 Difference of Means

The difference of means (DOM) compression method is based on the selection of samples approach, as mentioned above. The point selection is based on a pre-calculated signal strength estimate. The DOM method only considers the differences of values, and not the corresponding variances of the power traces, as essential for comparison – an important factor in our decision to implement it.

The DOM method is used to determine the relationship between the recorded power consumption matrix T (traces) and the columns of binary assumption-based matrix H. The H matrix is created by the attacker under the assumption that the power consumption for certain intermediate values is different for all other values; the binary value of H is a function of the input data d and a key hypothesis k_i, $h_{i,j} = HW(v_{i,j}), v_{i,j} = f(d_i, k_j)$. As suggested in [15], to reduce the HW model to a binary model, we set $h_{i,j} = 1$, if $HW(v_{i,j} \geq 4)$ and $h_{i,j} = 0$, if $HW(v_{i,j} < 4)$.

According to h_i, the attacker splits T into two sets of power traces (rows) for k_i's correctness check. The first set contains those T row indices corresponding to the indices of the zeros in the vector h_i, while vector m'_{0_i} denotes the mean of those rows $m_{0_{i,j}} = \frac{\sum_{l=1}^{n}(1-h_{l,j})\cdot t_{l,j}}{n_{0_i}}$. The second set contains all remaining rows in T, while vector m'_{1_i} is their mean vector $m_{0_{i,j}} = \frac{\sum_{l=1}^{n} h_{l,j}\cdot t_{l,j}}{n_{1_i}}$, where n denotes the number of rows in H and $n_{0_{i,j}} = \sum_{l=1}^{n}(1 - h_{l,i})$, $n_{1_{i,j}} = \sum_{l=1}^{n} h_{l,i}$.

A significant difference between the mean vectors m'_{0_i} and m'_{1_i} at some point in time indicates the correctness of key hypothesis k_i.

Each row in the results of matrix R: $R = M_1 - M_0$ corresponds to the differences between the mean vectors m'_{0_i} and m'_{1_i} of one key hypothesis.

3.3 Integration SNR

The integration methodology is a robust compression technique based on the selection of samples approach, which uses all of the recorded points in the power traces, and not just the peak/diff values. Unlike the DOM method, the integration SNR method takes the variance of the traces into consideration, an important factor in our decision to implement it.

The signal-to-noise ratio of a power sample is given by the following equation:

$$SNR = \frac{Var(P_{exp})}{Var(P_{sn} + P_{en})}$$

where P_{exp} is the exploitable power consumption, Psn is the switching noise, and Pen is the electronic noise. The SNR quantifies how much information is

leaking from a power trace. The higher the SNR, the greater the leakage. The integration of power trace in a time interval affects the SNR, since the signal and the noise of the recorded points in the time interval are summed. The SNR can be increased or decreased by the integration, depending on the time interval size used for the integration.

The time interval size is a decisive parameter for the compression's success; in cases in which there are many points with a strong signal in the time interval, the SNR will be high, and in cases where a single point is combined with points that leak little to no information (or no information at all), the SNR will be lower than the single point. Therefore choosing an appropriate time interval for the integration, like the length of a clock cycle (*cc*), is essential for a good compression process [15,24].

We implemented the following methods for integration-based power trace compression:

Integration Row (IR) computes the sum of all points of each time interval.

Sum of Absolute Values (SOA) computes the sum of the absolute values of all points of each time interval.

Finally, **Sum of Squares** (SOS) computes the sum of the squares of all points of each time interval.

We tested all methods with time interval ranges of [0.5, 1, 2] *cc*.

3.4 Top Score

The state-of-the-art profiling methods for this paper are based on the profiling methodology of [18]. In this work, Weisse et al. introduced a smart method for profiling the training set of power traces into an accurate decoder for an algebraic side-channel attack. The authors developed an efficient searching algorithm to identify the points in a trace that leak the most.

For the feature selection process, they proposed a scoring method for evaluating the amount of information each sample contains about a specific leak. Their profiling phase consists of the following steps:

1. Find regions of interest (ROI) in the traces for every leak using the Pearson correlation coefficient [4].
2. Calculate the feature scores for the features within the ROI of the evaluated leak identified in the previous step. The feature score is set as the average of 200 a-posteriori probabilities (of 200 evaluation traces) assigned to the correct Hamming weight by the Bayesian classier trained on the feature.
3. Create the best feature set (the set which contains the most information regarding the specific leak), which is used as input for the classier. Using the same Bayesian classifier and evaluation traces as the previous step, the mutual scores of all features in the best feature set are calculated; eventually, only the features that increase the score are included in the set. The best feature set size was statically limited to 500 features.

In our research, we use the success-rate scoring method of Weiss et al.'s by stabilizing the code from [29] and adjusting it as a compression method in the following way:

– Download the code base of [29]
– Parsing the RSM traces as describe in Sect. 2.2
– Mapping and isolating only the relevant code parts for the profiling and decoding analysis phases
– Adjusting the code base to matlab v2017 and rewriting respectively.

We denote this method as "Top Score" (TS). In our experiments, we limit the best feature set size to half the size of the training set.

3.5 Optimal Selection

Rechberger et al. [24] presented a practical TA using an efficient means of selecting the POI in a power trace and preprocessing noisy data. Their method is based on the maximum-extraction compression approach, in which the maximum peak values of the recorded samples in a clock cycle are simply extracted [15]. Instead of choosing the highest points, their advanced selection method defines a few properties that must exist at the selected points in the compressed trace:

– The minimum distance between the points should be approximately one clock cycle or more, since additional points in the same clock cycle do not provide additional information.
– The minimal height of a selected point should be higher than the noise floor of the sum of differences (SOD) trace.

In our research we implement the guidelines for the selection of POI from Rechberger's paper [24], as a substitute for operations of correlation calculation and scoring the features in the profiling phase. We adjust it as a compression method in our optimal suggested decoder, We call it "Optimal Selection" (OS). The term "optimal" was chosen due to its good trade-off between the performance parameters.

Three different minimum distances were chosen for testing; [0.5, 1, 2] *cc*. Not in accordance with Rechberger's recommendation, we also performed with 0.5 cc to test the selection of features in a situation where at the same clock cycle there are 2 samples at the same height (2 peaks).

The calculation of the noise floor is performed by multiplying the maximum value in the SOD traces with the noise factor which we set as 0.6 after testing the range of [0,1]. Unlike Rechberger, who considered constant numbers of POI ranging from 1 to 40 and set the level of the noise factor accordingly, we considered all the points which are higher than the noise floor calculated with the constant noise factor equal to 0.6, even if a higher number of points of interest is chosen.

4 Evaluation of Methods

We analyzed the performance of many TA profiling phase variants on real-world data, comparing all of the compression methods described in Sect. 3 using various configurations.

4.1 Evaluation System

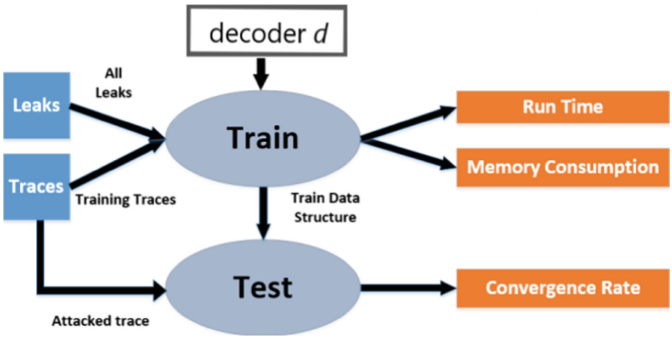


Fig. 1. Evaluation system architecture

The design of our evaluation system is a black-box for benchmark of a given decoder. The input of the system is decoder d, and the output is decoder d's training and test results for three parameters: runtime, memory consumption, and convergence rate. The system contains a data-set of traces and leaks for the training and test operation measurements. Our system architecture can be seen in 1. In our performance analysis we measured the following:

Memory Consumption: We measured each decoder's RAM usage (the total memory usage in bytes) as it processed the training set during the profiling phase. This was measured using the OS standard memory reporting function *vmstat* (Linux), when only the decoding process was running in our development environment.

Run-Time: We measured the operational runtime and overall convergence time of decoder as it processed the training set in the profiling phase. For the run-time measurements, we use the MATLAB timing functions *timeit,* to time how long the decoder code takes to run, and *tic-toc,* to measure the convergence time and operations' performance timing (the total execution time in seconds).

Convergence Rate: We estimated each decoder's quality based on its online convergence rate. This was measured by examining the number of traces that must be provided in the offline phase in order for each decoder to obtain reasonable results in the online phase. The convergence vector for each decoder is a Boolean vector representing the success/failure of the decoder's test results, with the overall convergence rate calculated as the mean of this vector.

4.2 Definition of the Training and Test Methods

The **training method** is used for model construction, classification and retention within its data structure; its input is pairs of (Trace T, Leak L). Training

consists of measuring run time and memory consumption and takes place in three steps: First, in the feature extraction step, a compression method is used. Next, in the feature pre-processing step, the selected features are grouped and labeled. Finally, in the template learning step, a naive Bayes classifier is trained for each leak (to distinguish between HW classes) using the features selected in the feature extraction step.

Preprocessing operations, such as leak calculation in Sect. 2.3 or trace parsing in Sect. 2.2, were not part of our measurements, since running them adds constant time which is not correlated with the decoding method.

The **test method** evaluates decoder d by receiving trace T (the attacked trace) as input, and examining its output, which consists of the HW of a certain leak based on its training process. This is validated on its data structure during testing. The validation results of *true* (success) or *false* (failure) are presented in the Boolean convergence vector of decoder d.

4.3 Experimental Setup

In our experiments, we explore profiled template attacks' feature selection (compression) methods for extracting the best subset of power samples for the classification of features according to the HW classes. All experiments are performed with MATLAB v2017, installed on an Intel Xeon E5-2620 CPU with 128 GB of RAM, running Ubuntu 18.04. The compression methods previously described are implemented to reduce the dimensions of the power traces. The subset sizes are selected based on the guidelines described in related studies. For each technique, we analyzed several configurations, and only the one yielding the best result is used in our overall comparison of the methods. Once the compressed trace is set, we implement the training and test methods (describe in Sect. 4.2) for each compression technique to evaluate its performance. We use 2,000 power traces from the initial dataset.

The steps of training and measuring performance are implemented in the following way:

We used Bayesian classifier for training template classifiers with HW leakage model for every leak. Using the *fitcnb* matlab function [16] which fits a naive Bayes classifier to data, we train a naive Bayes model on the selected features with the HW classes using normal distributions. We train a Bayesian classifier whose input features are those selected in the compression method. The training set consists of approximately half the number of traces examined in the experiment, and the other half serves as the evaluation set, in order to avoid an over-optimistic performance evaluation. The classifiers are trained to distinguish between Hamming weights 0–8. To measure the classification accuracy (%) of the trained classifiers during training, we used MATLAB's *predict* function [2]on the set of the evaluation traces (different from those used for training and testing), where the accuracy is the number of correctly classified leaks divided by the total number of leaks over the traces. To test the decoder, as described in Sect. 4.2 we used the classifier's data structure built during training and ran the *predict* function on the attacked trace taken from the other half of the examined

traces, which is denoted as the *test set*. We report the results of the performance measured during training and testing for each decoder.

5 Results

Tables 1 and 2 present the memory consumption (KB) and runtime (seconds) results during training. Table 3 presents the convergence rate (%) results for the testing phase. In each table, the row with the best configuration for each method appears in black (these results are included in our overall comparison). Figure 2 present the overall performance results for the compression methods examined. Table 1 presents the memory consumption results for each decoder during the training phase, with different amounts of training traces. Optimal Selection (OS) is the best performing compression method when considering memory usage; except for its low usage, identical values were obtained for training using different distances (*cc* values). Difference of Means (DOM) has the highest rates of memory consumption. Principal Component Analysis (PCA) follows, with 10 *pc* to keep found as the best performing setting for PCA. All SNR-based methods show the same trend in which the usage with 0.5 *cc* was higher than it was with 1 *cc*, and the usage was slightly lower with 2 *cc* than it was with 1 *cc*. The results obtained for all of the methods show that increasing the number of training traces results in higher memory consumption.

Table 2 presents the training phase's runtime results for each decoder, using different amounts of training traces. As seen in the table, TS has the longest run-times, and this is followed by DOM. Next comes OS and DOM, which shows identical values for different distances (*cc*). PCA follows with similar results found for 5 *pc* and 10 *pc*, while 30 *pc* shows longer results. The SNR methods had the shortest run-times, with Sum of Absolute Values (SOA) being the faster among these methods. In all methods we observed that increasing the distance by one *cc* cuts the runtime in half. The results for all methods show that as the number of training traces increases, the runtime also increases.

Table 3 presents the accuracy results for each decoder during testing phase, based on the number of traces in the training set. The "Top Score" (TS) method of [29] was found to be as the most accurate, and it is followed with OS, which has an accuracy greater than 70%; similar results were obtained for 1 *cc* and 2 *cc*. DOM also obtained good results, while PCA, which obtained the same results for all pc configurations, came next. The SNR methods obtained low results; while SOA performed the worst, the same trend was observed for all three SNR methods - increasing the distance by 1 *cc* significantly reduces the accuracy, sometimes by half. The results for all methods show that as the amount of training traces increases, the accuracy also increases.

Table 1. Memory consumption for all compression methods

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
DOM		2413736	3998680	4124432	4753541	4902672	5694248	6044480	6100464
PCA	5 *pc*	783445	850248	2088544	4268990	4562111	4672332	5091123	5100939
PCA	10 *pc*	772504	851496	2085968	4272824	4523424	4609552	4961832	5139784
PCA	30 *pc*	1116992	1206768	2406654	4466343	4809202	4999765	5163442	5432887
TS		338160	954512	1182768	2180192	2749328	3894856	4733248	4812720
IR	0.5 *cc*	281396	4011232	4088761	4298778	4311023	4695341	4810116	4955809
IR	1 *cc*	182304	2931551	3116088	3353981	3385451	3722401	3774523	3911296
IR	2 *cc*	175772	2864598	3108973	3294332	3334423	3566878	3672445	3668915
SOA	0.5 *cc*	190203	1973771	2154481	3700231	4378922	4399872	4752112	4998213
SOA	1 *cc*	111408	1353984	1404264	2856560	3421216	3554512	3856481	4131128
SOA	2 *cc*	100897	1184562	1302213	2671333	3302451	3267763	3676884	4102893
SOS	0.5 *cc*	278965	338767	387291	3909845	4144521	4453201	5022318	5296659
SOS	1 *cc*	199432	442704	290176	3014256	3271528	3661376	4149342	4350296
SOS	2 *cc*	190567	240561	264367	2889416	3240567	3653299	4103292	4203389
OS	0.5 *cc*	96236	132271	280038	279913	450221	718134	841761	1125032
OS	1 *cc*	96160	130488	281840	282224	449848	715920	839288	1137648
OS	2 *cc*	95988	129850	282098	290013	451211	719656	836114	1135451

Table 2. Run-Time for all compression methods using various

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
TS		617.2	964	995.2	1316.1	2037.3	2297.3	2418.9	2428
DOM		506.9	570.3	598.4	610.2	703.6	720.3	747.2	818.3
OS	0.5 *cc*	153.1	264	302.9	323.7	384.2	442.5	497.1	525.9
OS	1 *cc*	152.4	262.3	301.5	323.8	383.2	443.6	495.4	525.8
OS	2 *cc*	152.2	263.7	301.8	324.1	383.1	442.6	496.1	523.9
PCA	5 *pc*	91.2	173.1	211	235.4	289.5	325.9	353.2	387.8
PCA	10 *pc*	91.3	173.2	211.2	235.7	289.9	326.7	353.9	388.6
PCA	30 *pc*	102.5	181.9	220.7	248.1	301.4	339.8	377.8	405.2
IR	0.5 *cc*	78.2	91.3	168.9	230.8	257.8	293.6	310.3	341.8
IR	1 *cc*	37.1	46.8	85.6	115.5	130.4	145	156.6	171.9
IR	2 *cc*	16.2	23.6	43.4	59.1	66.6	74.1	79.8	85.2
SOS	0.5 cc	128.2	132.3	170.6	176.2	343.5	380.7	403.2	438.9
SOS	1 *cc*	65.9	68.7	87.2	139.4	174.2	192.2	206.8	222.8
SOS	2 *cc*	32.7	36.4	44.8	68.1	89.5	97.1	102.5	110.1
SOA	0.5 *cc*	92.5	132.7	177.5	236.8	264.2	280.6	361.7	431.2
SOA	1 *cc*	47.2	67.6	90.3	119	133.1	144.3	182.5	218.1
SOA	2 *cc*	25.8	34.8	47.1	62.6	66.6	73.5	94.7	111.3

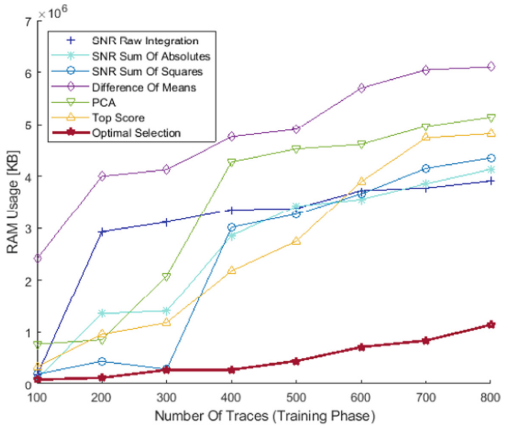
Table 3. Convergence rate for all compression methods

Method	Config	100T	200T	300T	400T	500T	600T	700T	800T
TS		0.64	0.66	0.7	0.71	0.71	0.724	0.75	0.785
OS	0.5 *cc*	0.44	0.46	0.48	0.5	0.5	0.52	0.552	0.56
OS	1 *cc*	0.52	0.57	0.61	0.636	0.65	0.653	0.68	0.72
OS	2 *cc*	0.52	0.57	0.61	0.635	0.65	0.65	0.68	0.72
DOM		0.55	0.56	0.58	0.632	0.64	0.642	0.673	0.71
PCA	5 *pc*	0.2	0.24	0.24	0.23	0.25	0.257	0.271	0.29
PCA	10 *pc*	0.22	0.24	0.24	0.246	0.252	0.26	0.274	0.3
PCA	30 *pc*	0.22	0.241	0.244	0.248	0.255	0.26	0.27	0.3
IR	0.5 *cc*	0.06	0.07	0.07	0.08	0.09	0.092	0.092	0.097
IR	1 *cc*	0.18	0.19	0.2	0.2	0.204	0.21	0.219	0.22
IR	2 *cc*	0.1	0.11	0.114	0.114	0.115	0.117	0.12	0.126
SOS	0.5 *cc*	0.04	0.06	0.1	0.1	0.105	0.114	0.14	0.146
SOS	1 *cc*	0.11	0.145	0.17	0.18	0.185	0.2	0.2	0.21
SOS	2 *cc*	0.07	0.1	0.123	0.13	0.128	0.14	0.144	0.16
SOA	0.5 *cc*	0.03	0.03	0.03	0.034	0.04	0.04	0.041	0.047
SOA	1 *cc*	0.11	0.13	0.15	0.15	0.16	0.166	0.17	0.19
SOA	2 *cc*	0.06	0.08	0.09	0.09	0.09	0.098	0.1	0.1

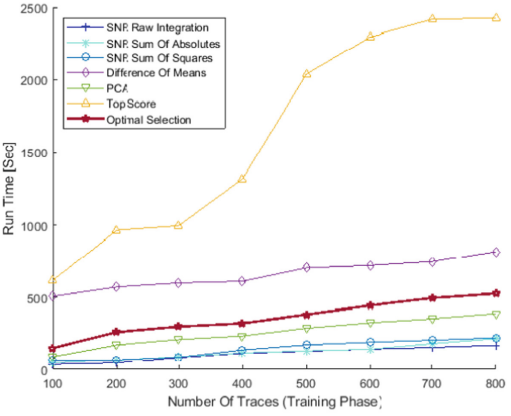
5.1 Observations

We now compare the decoder's best performing configurations for memory consumption, runtime, and convergence rate and make some general observations based on this comparison.

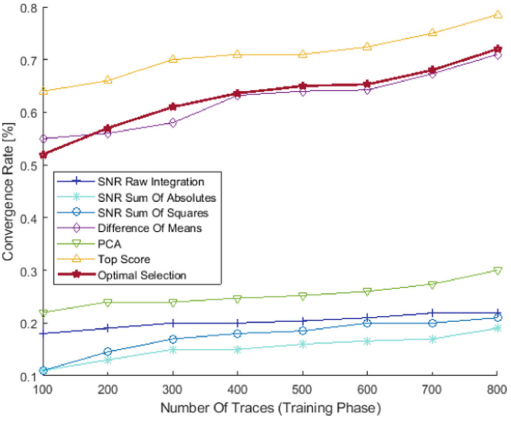
- Testing 0.5 *cc* as a minimum distance between the selected features presented in all measurements of all characteristics provides identical or worse results to the 1 *cc* configuration.
- The 10 *pc* configuration was determined to be the best PCA configuration, since, on average, it has the same runtime as the 5 *pc* configuration, but this is obtained with lower memory consumption. On the other hand, the 10 *pc* configuration has identical accuracy to the 30 *pc* configuration, but the runtime and memory consumption of the 10 *pc* configuration are significantly shorter and lower, respectively, than that of the 30 *pc* configuration.
- For the SNR methods, we determined that the 1 *cc* configuration was the most accurate; its runtime was half that of the 0.5 *cc* configuration, and it also had much lower memory consumption; when compared to 30 *pc*, the 1 *cc* configuration had significantly shorter run-times and lower memory consumption.
- OS has identical runtime and memory consumption results for the various configurations; in terms of accuracy, 1 *cc* and 2 *cc* were found to be identical, so the configuration of 1 *cc* was selected for our overall comparison.



(a) Memory consumption - overall comparison



(b) Run Times - overall comparison



(c) Convergence rate - overall comparison

Fig. 2. Overall comparison

Figure 2 presents a comparison of the performance of all of the compression methods, using the best configuration for each method. Based on this comparison, we can see that TS is never the best method in terms of runtime (see Fig. 2b), and the same can be said for DOM in terms of memory consumption (see Fig. 2a in the Appendix). The SNR-based methods show the same performance trend for all measures, where in most cases, IR is the best performing of these methods and SOA is the worst performing. The PCA method was never abnormally slow or poor performing, but it was always outperformed by at least one of the other methods we compared in this study. Optimal selection (OS), our proposed compression method, had the lowest memory consumption of all of the decoders (see Fig. 2a in the Appendix) and the shortest runtime of the three decoders with the highest accuracy (see Fig. 2c). After TS, it was found to be most accurate with a 72% convergence rate given only 800 traces used in the training phase (see Fig. 2c).

6 Conclusion

In this paper, we strove to advance the profiling step of the template attack, by seeking a practical compression method which requires a smaller dataset and has better performance, both in the online and offline phases. We addressed the challenge of finding the most informative traces regarding the leaked Hamming weight values by building an optimal variant of the state-of-the-art decoder presented in [18], based on the optimal feature selection guidelines of [24].

For the performance challenge, we designed a unique evaluation system which measured runtime, memory consumption, and accuracy. This system used to compare the performance of various decoders, which were found to differ based on the compression methods and configurations used. We demonstrated the importance of both choosing the correct number of principal components for the PCA-based method, and the correct number of clock cycles as a minimum distance between the selected points; one clock cycle was clearly found to be optimal in all of the best performing configurations. In terms of accuracy, the scoring, optimal, and DOM methods outperformed the PCA and SNR-based methods. When considering runtime and memory the opposite is true, except in case of our OS decoder.

The experimental results demonstrate our decoder's ability to outperform the other methods evaluated in terms of memory consumption; however, while it also has shorter run-times than the state of the art, it is slightly less accurate. There is thus a tradeoff in that while our decoder is fast and has low memory consumption, this comes at a cost in terms of the accuracy rate; therefore, our decoder is optimal in cases in which there are time or data restrictions, for example, a small dataset or online data.

References

1. MATLAB PCA. https://www.mathworks.com/help/stats/pca.html
2. MATLAB predict. https://in.mathworks.com/help/ident/ref/predict.html
3. Bohy, L., Neve, M., Samyde, D., Quisquater, J.J.: Principal and independent component analysis for crypto-systems with hardware unmasked units (2003)
4. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
5. Cagli, E.: Feature extraction for side-channel attacks. Ph.D. thesis, Sorbonne University, France (2018)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
7. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 253–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_17
8. Division, C.S.: Announcing the Advanced Encryption Standard (AES). Information Technology Laboratory, Gaithersburg, MD (2001)
9. Elaabid, M.A., Guilley, S.: Practical improvements of profiled side-channel attacks on a hardware crypto-accelerator. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 243–260. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12678-9_15
10. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 15–29. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_2
11. Hogenboom, J.: Principal component analysis and side-channel attacks (2010)
12. Kay, S.M.: Fundamentals of Statistical Signal Processing: Estimation Theory. Signal Processing Series, 1st edn. (1998)
13. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
14. Renauld, M., Standaert, F.-X.: Algebraic side-channel attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16342-5_29
15. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smart Cards. Springer, Cham (2007). https://doi.org/10.1007/978-0-387-38162-6. ISBN 978-0-387-30857-9
16. MathWork: MATLAB fitcnb. https://in.mathworks.com/help/stats/fitcnb.html
17. Oren, Y., Renauld, M., Standaert, F.-X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 140–154. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_9
18. Oren, Y., Weisse, O., Wool, A.: Practical template-algebraic side channel attacks with extremely low data complexity. In: HASP@ISCA, p. 7. ACM (2013)
19. Oswald, D., Paar, C.: Improving side-channel analysis with optimal linear transforms. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 219–233. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37288-9_15
20. ParisTec: DPA contest v4 2013. http://www.dpacontest.org/v4/rsm_traces.php

21. ParisTec: Description of the masked AES - DPA contest v4 (2013). http://www.dpacontest.org/v4/data/rsm/aes-rsm.pdf
22. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. J. Cryptogr. Eng. **7**(4), 343–351 (2017). https://doi.org/10.1007/s13389-017-0172-7
23. Picek, S., Heuser, A., Jovic, A., Batina, L.: A systematic evaluation of profiling through focused feature selection. IEEE Trans. Very Large Scale Integr. Syst. **27**(12), 2802–2815 (2019)
24. Rechberger, C., Oswald, E.: Practical template attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 440–456. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31815-6_35
25. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_8
26. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting time samples for multivariate DPA attacks. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 155–174. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_10
27. Stallings, W.: Cryptography and Network Security, 6th edn. (2014)
28. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Profiling attack using multivariate regression analysis. IEICE Electron. Express **7**(15), 1139–1144 (2010)
29. Weiss, O.: Github - new methods for side channel cryptanalysis code base github (2016). https://github.com/oweisse/dpav4-contest/commits/master
30. Weisse, O.: New methods for side channel cryptanalysis (2013)
31. Zheng, Y., Zhou, Y., Yu, Z., Hu, C., Zhang, H.: How to compare selections of points of interest for side-channel distinguishers in practice? In: Hui, L.C.K., Qing, S.H., Shi, E., Yiu, S.M. (eds.) ICICS 2014. LNCS, vol. 8958, pp. 200–214. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21966-0_15