Reality Check on Side-Channels: Lessons learnt from breaking AES on an ARM Cortex A processor

Shivam Bhasin
sbhasin@ntu.edu.sg
Temasek Labs @ NTU
Singapore

Harishma Boyapally
harishma.boyapally@ntu.edu.sg
Temasek Labs @ NTU
Singapore

Dirmanto Jap
djap@ntu.edu.sg
Temasek Labs @ NTU
Singapore

ABSTRACT

AES implementation has been vastly analysed against side-channel attacks in the last two decades particularly targeting resource-constrained microcontrollers. Still, less research has been conducted on AES implementations on advanced hardware platforms. In this study, we examine the resilience of AES on an ARM Cortex A72 processor within the Raspberry Pi 4B model. Unlike their microcontroller counterparts, these platforms operate within the complex ecosystem of an operating system (OS), resulting in EM traces characterized by low signal-to-noise ratios and jitter. We discuss the inefficacy of traditional CPA attacks in the presence of noise, misalignment, and jitter (in trace and trigger signals). The interrupts and daemons cause these effects, resulting in context switch overheads leading to increased variability in execution times. Additionally, there are no fixed methods or set rules for pre-processing; the approach varies depending on the target device. Our experiments show that CPA is ineffective against masked and unmasked AES implementations on ARM Cortex A72. Therefore, we resort to deep learning-based side-channel analysis (DL-SCA) techniques, that do not require extensive data pre-processing and can effectively work with EM traces that have low signal-to-noise ratios. Using DL-SCA we could recover the AES secret key. Our experiments underscore the formidable challenge posed by breaking AES on ARM Cortex processors compared to conventional microcontroller-based implementations. Importantly, our findings extend beyond previous studies, marking the first successful attack on ARM Cortex A72 and demonstrating the efficacy of DL-SCA even when pre-processing techniques are varied and not standardized.

KEYWORDS

AES, Complex Processors, EM Side-channel, Deep Learning based Side-Channels, Raspberry Pi, Arm Cortex A72

1 INTRODUCTION

AES (Advanced Encryption Standard) [26] is a symmetric encryption algorithm, vital for upholding data confidentiality and security. It has been extensively explored in literature and has undergone rigorous scrutiny against traditional cryptanalysis and side-channel vulnerabilities on microcontrollers and FPGAs. Correspondingly, several countermeasures are proposed, to be incorporated as part of software and hardware implementations. In recent years, devices such as smartphones, tablets, automotive systems, and medical devices have been equipped with more advanced processors, including the ARM Cortex M and A series. They perform complex and resource-intensive operations to provide a wide range of functionalities. However, there is limited research conducted on such platforms.

In [6] authors present DPA attacks on a bitsliced implementation of AES running on a 1 GHz ARM Cortex-A8 processor with OS in the background. Jauvart et. al [17] analysed side-channel vulnerabilities on ARM Cortex-M3 and demonstrated that standard cryptographic pairings are susceptible to SCAs, highlighting the need for optimized countermeasures to minimize leakage. In [29], the authors employed machine learning to build a side-channel disassembler for the ARM-Cortex M0. They achieved a high success rate in distinguishing between instruction groups, demonstrating the feasibility of using side-channel data to deduce executed instructions. A correlation power attack on the AES-128 encryption algorithm of the Crypto++ library has been shown in [14]. The attack is conducted on ARM cortex-A8, modifying the frequency from 1GHz to 600 MHz. Though they conduct side channel analysis on ARM Cortex M and A series, they are old models and in recent years more advanced processors have been in the market.

In this study, we focus on the security of AES implementations on the more intricate ARM Cortex-A 72 device on the Raspberry Pi platform with full OS running in the background. The existing literature suggests that AES is vulnerable to side-channel leakage. However, in this work, we aim to explore the magnitude of difficulty in mounting such an attack on complex processors. Our investigations begin by evaluating the trace quality using standard leakage assessment methods and probing the feasibility of recovering the correct key in known key settings. Raspberry Pi, as expected, proved to be a difficult target to attack, where the electromagnetic (EM) traces collected during the AES run are shown to have low SNR and high misalignments. In the absence of concrete methods to filter out noise and obtain traces with leaky sample points, we resort to ad-hoc pre-processing techniques. However, these techniques are insufficient to successfully employ correlation power analysis (CPA). Deep neural network-based side channel attacks [15, 19, 20, 25] have shown promising results without the need for preprocessing techniques. The extensive range of hyperparameters enhances the performance of side-channel attacks on AES implementations, even when strong countermeasures are in place.

Our contributions in this paper are twofold:

(1) We present an evaluation of AES implementation in the practical setting with real devices. Due to the environmental noise from the full-fledged OS running in the background, classical CPA has been proven ineffective against AES on ARM Cortex A72. We demonstrate the capability of DL-SCA to achieve successful key extraction without the need for preprocessing, an instance of DL-SCA employed in attacking a real-world target.

(2) The AES dataset generated in our study proves invaluable for further investigation, particularly in the realm of DL-SCA. While existing public datasets such as ChipWhisperer [23], DPAv4.2 [2], AES RD [1], AES HD [9], ASCAD [4, 7], and CHES CTF [3] exist, many of them remain vulnerable to classical attacks. Our dataset[1] offers enhanced explainability for DL-SCA, addressing the need for a novel dataset in this domain.

2 BACKGROUND

In this section, we lay the groundwork for the approaches that will be employed throughout the paper. Firstly, we introduce the basic concept of side-channel attacks (SCA). Next, we discuss leakage assessment fundamentals, followed by an overview of non-profiling and profiling-based attacks. Finally, we delve into the explanation of deep-learning-based SCA.

2.1 Side-Channel Attacks (SCA)

Side-channel attacks, abbreviated as SCA, represent a significant threat to the physical implementation of cryptographic algorithms. Despite the theoretical security of cryptographic algorithms, their physical executions can inadvertently leak information through various physical channels, such as power consumption [18] and electromagnetic emanation [5]. These unintentional disclosures, often termed as side-channel traces or simply traces, are then analyzed to recover secret data.

The two most common types of attacks are non-profiling and profiling attacks. In non-profiling attacks, attackers utilize statistical analysis to exploit the traces and recover secret data. Conversely, profiling attacks involve attackers having access to a device similar to the target device, commonly known as a clone device, enabling them to construct a leakage profile by fully accessing the device. This leakage profile is then utilized by the attackers to recover secret data from the actual target device.

2.2 Leakage Assessment

To assess the quality of the measured traces, Test Vector Leakage Assessment (TVLA) [16] can be employed. The principle behind TVLA involves using statistical tests to observe differences in distribution between two datasets. The standard approach entails conducting a Welch t-test on datasets obtained from sending fixed and random inputs (denoted as x and y, respectively):

$$t = \frac{\overline{x} - \overline{y}}{\sqrt{\frac{\sigma_x^2}{N_x} + \frac{\sigma_y^2}{N_y}}},$$

where $\overline{x}$, $\overline{y}$, σ_x^2, and σ_y^2 represent the mean and variance of x_i and y_i, respectively.

If the absolute values of t, referred to as $|t|$-values, exceed the threshold of 4.5, it indicates, with 99.999% confidence, the presence of data-dependent leakage in the measured traces. Such findings may suggest potential leakages in the implementation.

[1]We will publish the full dataset upon acceptance.

2.3 Non-Profiling Attacks

One of the most commonly utilized non-profiling attacks is Correlation Power Analysis (CPA) [10]. The essence of this approach lies in employing Pearson correlation to ascertain the relationship between intermediate values computed from various secret hypotheses and the actual leakage values. Attackers leverage intermediate values computed as a function f of known inputs p and (hypothetical) secret $k \in K$. Specifically, attackers compute $H = f(p, k)$. Subsequently, these intermediate values are compared with the actual leakage traces T obtained while processing the genuine secret k^*. The secret k exhibiting the highest (absolute) correlation can then be estimated as the true secret value.

The Pearson correlation between x and y can be computed as follow:

$$r(x, y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \quad (1)$$

To correlate intermediate values with proper leakage, they must first be mapped using an appropriate leakage model tailored to the target device. Typically, a leakage model is employed to approximate the behaviour of measured traces. For software implementations, the leakage is often assumed to follow the Hamming weight (HW) model. Conversely, for hardware implementations, the Hamming distance (HD) model is commonly utilized. Additionally, an identity mapping (ID) can serve as an alternative leakage model.

2.4 Profiling Attacks

Profiling attacks operate under the assumption of a worst-case scenario where attackers possess access to a clone device identical to the target device. With full control over this cloned device, attackers can acquire knowledge (referred to as a profile) to exploit the actual target device. In this scenario, attackers may manipulate or obtain the key of the clone device, while the key of the target device remains unknown to them. Additionally, attackers have the capability to collect multiple traces from a known set of random plaintexts (or ciphertexts) from both devices.

Initially, attackers obtain profiling traces from the clone device, subsequently acquiring attack traces from the target device for exploitation. The objective of attackers is to recover the unknown secret key from the target device. One of the most prevalent profiling attacks is Template Attacks (TA) [12, 13]

As explained, profiling attacks can be delineated into two phases: the profiling phase and the attack phase. During the profiling phase, a distinguisher $\mathcal{F}$ is constructed from the set of profiling traces. This distinguisher yields a conditional probability mass function $\Pr(\boldsymbol{T}|Z = z)$. In the attack phase, the distinguisher returns a probability score for each hypothetical sensitive value. Specifically, we obtain $\boldsymbol{y}_i = \mathcal{F}(\boldsymbol{t}_i)$, where $\boldsymbol{t}i$ represents an attack trace. The log-likelihood score for each key $k \in \mathcal{K}$ is computed as follows:

$$s_{N_a}(k) = \sum_{i=1}^{N_a} \log(\boldsymbol{y}_i[z_{i,k}]),$$

where N_a denotes the number of attack traces used and $zi, k = C(p_i, k)$ represents the hypothetical sensitive values based on key k, with p_i being the corresponding public variable to trace $\boldsymbol{t}_i$, and C denoting the cryptographic primitive.

Subsequently, the keys are ranked based on their log-likelihood scores in descending order and classified into a guess vector $\boldsymbol{G} = [G_0, G_1, \ldots, G_{|\mathcal{K}|-1}]$, where G_0 corresponds to the score of the most likely key candidate, and $G_{|\mathcal{K}|-1}$ represents the score for the least likely key candidate. The rank of each key is denoted by its index in the guess vector $\boldsymbol{G}$. The guessing entropy GE [27] is defined as the average rank of the correct key k^* across multiple experiments. A successful attack is achieved when $GE = 0$, indicating that N_a attack traces are sufficient. We denote $NTGE$ as the minimum number of traces required to achieve $GE = 0$.

2.5 Deep Learning-based Side-Channel Attacks (DL-SCA)

Recent reports indicate that Deep Learning (DL)-based approaches have surpassed classical profiling-based attacks [11, 21, 28]. DL-based Side-Channel Analysis (DL-SCA) offers several advantages. Firstly, it effectively handles jitter or de-synchronization in traces owing to its shift-invariant property [11]. Additionally, DL-SCA can accommodate masked implementations. Traditional SCA requires combining time samples where the mask and the masked value are processed, posing challenges in identifying precise time samples and complicating matters further with trace de-synchronization. DL-SCA, however, automates the combination of these samples during training and can directly target masking implementations [22].

In this study, we adopt a similar approach, employing Convolutional Neural Networks (CNNs), a commonly used learning paradigm in DL.

CNNs typically comprise three types of layers: convolutional, pooling, and fully connected layers. The convolution layer computes the output of neurons connected to local regions in the input, with each neuron computing a dot product between the weights and input within a small region (convolution kernel). The pooling layer downsamples the number of features by combining outputs from a group of neurons in the previous layer into a single neuron in the next layer, usually by taking the max or average. Finally, in fully connected layers, every neuron in one layer is connected to every neuron in another layer.

2.5.1 Random Search Hyperparameters for CNN. As optimizing the CNN model is not the focus of this study, we did not delve into further investigations for its optimization. Instead, we opted for hyperparameter tuning using random search. This method has been employed previously [24, 30, 31]. The concept involves defining a set of parameters and creating multiple models with randomly selected parameters from the set. These models are then trained with the random parameters on the training data, and their performances are evaluated on the validation data. Subsequently, the best-performing models are selected and deployed for targeting the attack traces.

In Table 1, we provide the parameter spaces that we use for random hyperparameter search. Note that in this work, we do not aim to find the most optimal model, however, we show that it is possible to design a network that will allow successful key recovery, and as such the study on how to find the most optimal network will be left for future works.

Parameters	Range
Batch Size	100, 200, ... 1000
Learning Rate	1e-3, 5e-4, 1e-4, 5e-5, 1e-5
Optimizer	RMSprop, Adam
Layers	1,2,...,8
Neurons	10,20,50,100,200,300,400,500
Kernel Initializers	{random,glorot,he}_uniform
Pooling	{max,average}_pool
Pooling size	2,4,6,8,10
Convolution layer	1,2,3,4
Filters	4,8,12,16
Kernels	26, 28, ..., 52
Padding	0, 4, ..., 16

Table 1: Range for hyperparameter random search

Algorithm 1 Single S-box Operation with Sleep Operations

```
Sleep()
x ← plaintext ⊕ key_byte
ciphertext ← S-box[x]
Sleep()
```

3 EXPERIMENTS

For experiments, we use the Raspberry Pi Model 4B board featuring the ARM Cortex-A72 processor as our designated platform. To collect the side-channel traces, we used the high-sensitivity Riscure EM probe (ZVC45 series) with built-in amplifier, measured on the LeCroy oscilloscope. We also used a high-pass filter.

One of the initial tasks is to find the location for EM probe placement to get side-channel traces with observable leakage. Finding optimal placement can be quite challenging. We considered areas near the processor and the decoupling capacitors on both the top and bottom sides of the board to obtain useful traces.

We implemented AES-128 in C language, on the Raspberry Pi, and collected a set of traces, for each designated probe position on the board. Employing various methodologies, including TVLA assessments with fixed versus random key differentials and CPA analysis, we scrutinized the traces for discernible patterns indicative of cryptographic leakage. We note that, while performing our experiments, a full OS is running on the board. In this work, we discuss the challenges encountered and potential mitigation.

3.1 Challenges

Our first goal is to find an appropriate position on the board, that can leak data-dependent side channel information. We run the code snippet described in Algorithm 1, focusing only on the trace part containing the AES S-box operation. The corresponding TVLA plot at one probe position is shown in Figure 1, indicating the presence of leakage. We noted the positions with potential leakage. We can see that there are several sample points corresponding to the S-box operation, that result in TVLA results being less than -4.5 and greater than 4.5. This implies that there is in fact leakage in the locations. However, as TVLA only tells us whether there is leakage

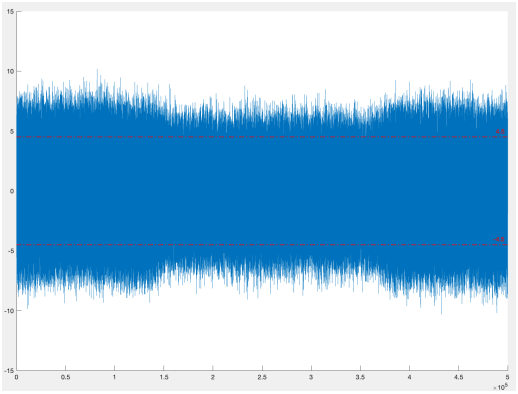


Figure 1: TVLA for Algorithm 1 on 100K traces and 500K sample points. The dashed lines indicate the threshold [-4.5,4.5].

or not, but not whether it is exploitable, we still have to proceed first with a key recovery attack.

Despite this approach, we could not perform successful key recovery (as shown in Figure 2). Notably, the presence of low signal-to-noise ratio (SNR) and consequential jitter induced by background operating system activities posed considerable obstacles. We cannot conclude whether the position of the probe is non-leaky or if the noise is too high to retrieve the key. Additionally, we also observed that the TVLA corresponding to the noisy operations have good TVLA values, and even higher than AES. This implies that, while the sleep or NOP operations are being executed, several background activities are corresponding to the OS, scheduling, interrupts, daemons etc, that are constantly executing and leaking information, that is captured. Additionally, we can also note that ARM Cortex A72 supports multi core functionality on the Raspberry Pi board. This implies that, even while AES is running, the EM signals not only correspond to AES but to the background operations. This results in low SNR traces, that cannot help with the CPA attack to retrieve secret key bytes.

Upon careful observation of the traces, we noticed that there is jitter in the traces and trigger. To avoid the loss of useful sample points, we fixed the rising edge of the trigger and added sleep operations after the AES implementation. This ensures that the jitter on the falling edge does not result in the loss of useful sample points in the trace signal. To find a better location, we remove the metal casing on the processor, ensuring no damage or disruption to the device. Figure 4 shows the position of the Riscure EM probe on the RPi board, after removing the metal casing on the processor.

To isolate the noise from the target AES operation, we integrated deliberate sleep operations preceding and following AES computations, essentially making our algorithm as shown in Algorithm 2 for a fixed k (here $k = 5$). We visually monitored the trace shape and obtained a possible leaky position as shown in Figure 4 on top of the processor. The resulting trace is shown in Figure 3. We then run a single S-box implementation, with NOP operations, before and after. The corresponding traces produce TVLA leakage, but when we performed a CPA attack on 2K traces and 500K sample points in each trace, it was to no avail as shown in Figure 5. Then we went back to the implementation Algorithm 2 and varied $k \in [1, 5]$.

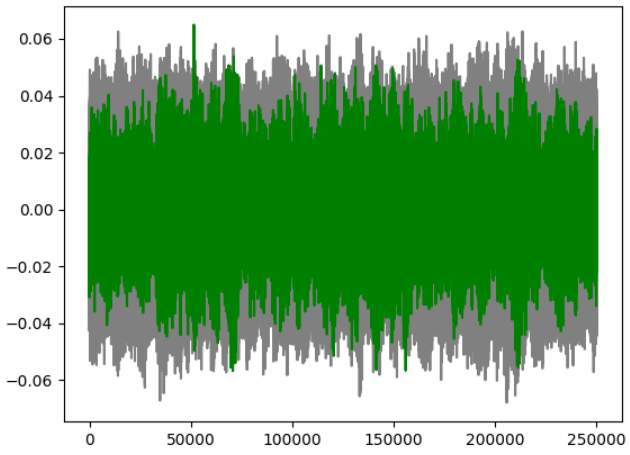


Figure 2: Unsuccessful CPA attack, to retrieve first key byte, with 250K sample points. The grey colour indicates an overlapping correlation for each sample point, for the wrong key guess and the green colour indicates the correlation for the correct key byte guess.

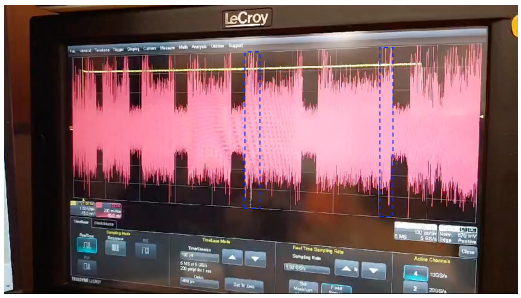


Figure 3: EM Trace for Algorithm 2 with $k = 5$ and sampling rate 5GS/s.

We noticed that the trace length is not proportional to the sleep period but to the number of sleep operations. We attribute this to the context switching performed by the operation during the transition from sleep to the AES operations (as marked inside the blue dashed box).

Algorithm 2 Algorithm to find suitable position for trace collection

```
1: Sleep()
2: for i ← 1 to k do
3:     AES-128()
4:     Sleep()
5:     for j ← 1 to i do
6:         AES-128()
7:     end for
8: end for
9: Sleep()
```

We again attribute this limitation to the complex interplay of various factors from the target device, such as signal integrity, noise interference, and jitter in the traces. The traces not only are

Figure 4: Raspberry Pi board with EM probe positioned on the ARM Cortex A72 Processor

being influenced by the background operations, they are also very misaligned due to the jitter in trace and trigger. We further analyze the traces to notice that as confirmed by the trigger length, the time taken to compute the S-box operation keeps changing due to these challenges. In Figure 6, we show a contour plot for these triggers corresponding to 1,000 traces and 2,000 sample points.

We can see the falling edge of the trigger extensively moving in the range of $[800, 1400]$. We can notice some triggers crossing the capture range and some falling in the range of $[200, 600]$. This implies that the sample points for S-box computation move along the trace, which will undoubtedly fail CPA. This also implies that the TVLA leakage observed in Figure 1, is not a result of the data-dependent leakage of AES.

We have also tried the standard realignment method, such as Dynamic Time Warping (DTW) [8]. However, it is not working as well, since there is a lack of reference traces, and the computational complexity is too costly in this scenario. As such, we use the following preprocessing method, due to its simplicity, to select a subset of traces and discard the unwanted ones.

3.2 Preprocessing

As described previously, we observed de-synchronization from the traces, evident from the varying trigger lengths observed for a single S-box operation. To address this issue, we initiated a preliminary preprocessing phase to refine the traces. Our approach is as follows:

(1) Examination of Trigger Lengths: We began by scrutinizing the lengths of the triggers.
(2) Distribution Plotting: Subsequently, we plotted the distribution of trigger lengths.
(3) Mode Identification: From the distribution, we identified the mode of the trigger length.
(4) Trace Selection: Traces exhibiting trigger lengths akin to the mode were selected, constituting approximately 8-18% of the total collected traces.

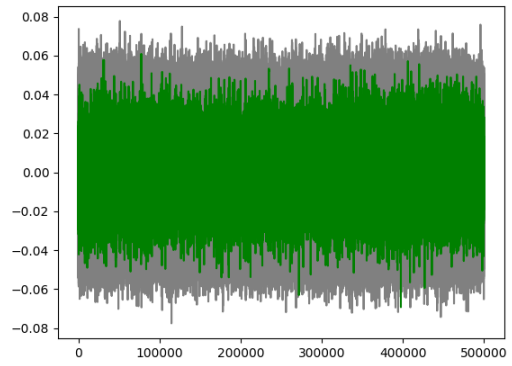


Figure 5: Unsuccessful CPA attack with the metal casing removed on the board, to retrieve first key byte, with 500K traces (grey refers to the wrong key guesses and green refers to correct guess)

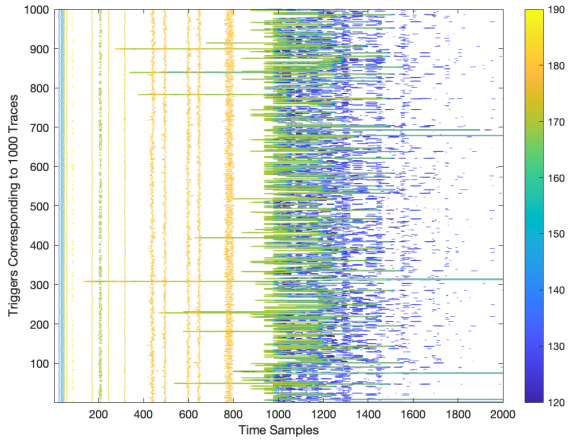


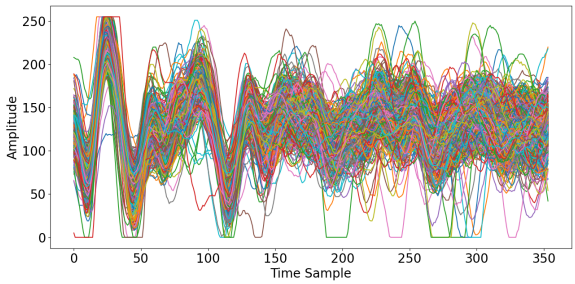
Figure 6: A contour plot representing 1000 triggers with fixed rising edge corresponding to the traces of a single S-box operation without the sleep operation.

These steps will result in traces with better alignment.

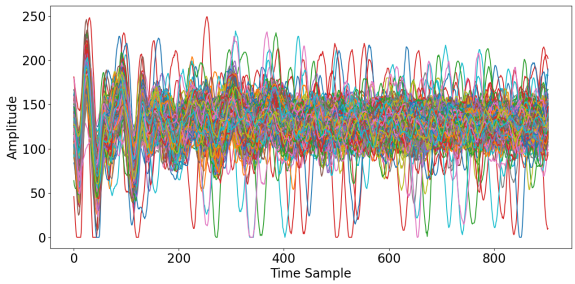
3.3 Leakage Assessment Using TVLA

With the above observations, we implemented a single S-box operation with fixed and random key bytes on unprotected AES C implementation and collected the EM traces. We do this experiment on O0 implementation (implying no optimization), and the traces are preprocessed.

First, we plot and observe the shape of the traces we have collected. In Figure 7, we plotted the first $1,000$ traces obtained for unprotected and masked implementation. For masked implementation, we implemented the masked version used in the ASCAD dataset ([7], Algorithm 1) and we targeted the AddRoundKeys and SubBytes operation (lines 6-8). As can be observed from the plots, there is de-sychronization, in the sense that the traces are not fully aligned. We also observed that there are shifts in the amplitude (due to the signal being affected by background processes), as can be observed quite clearly in Figure 7a, which results in the traces getting



(a) First 1, 000 traces for unprotected AES



(b) First 1, 000 traces for masked AES

Figure 7: Traces with random keys

clipped. These indicate that the environmental noise is not only causing the de-synchronization but also affecting the leakages. We make similar observations for the case of masked implementation in Figure 7b.

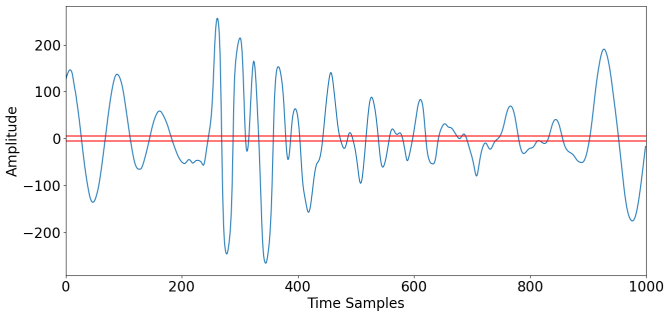
We then proceed with TVLA to assess the quality of the leakage. Encouragingly, these analyses confirmed the presence of leakage in the identified position, affirming our hypothesis, as can be observed in Figure 8a. This process is done using 300,000 traces, alternating between fixed and random input to the AES.

As can be seen from Figure 8b for masked AES implementation, unlike the one observed from Figure 8a, even though there are several points exceeding the threshold, it leaks less than unprotected AES C implementation, which is expected for the countermeasure protected implementation. Next, we proceed to perform CPA analysis on these preprocessed traces.

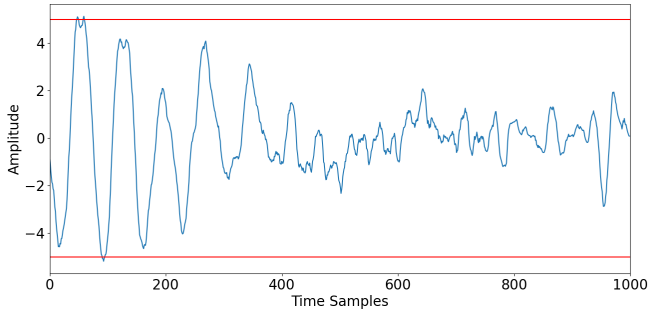
3.4 Analysis using CPA (non-profiled)

We conducted CPA on both protected and unprotected traces (300, 000 traces). First, we investigated the unprotected traces and we tried to recover one byte of the secret key. After the CPA experiment, we observed that the secret key could not be recovered, as shown in Figure 9. Here, we can safely assume that de-synchronization in the traces plays a major role in preventing univariate attacks like CPA. This also highlights that even though the TVLA test clearly indicates potential leakages, it might not be necessarily exploitable.

In the next phase, we performed CPA on masked implementation, assuming the knowledge of the mask. In this case, we assume the role of evaluator who knows the random mask value and uses this knowledge to mount the attack to recover the secret key. Notice

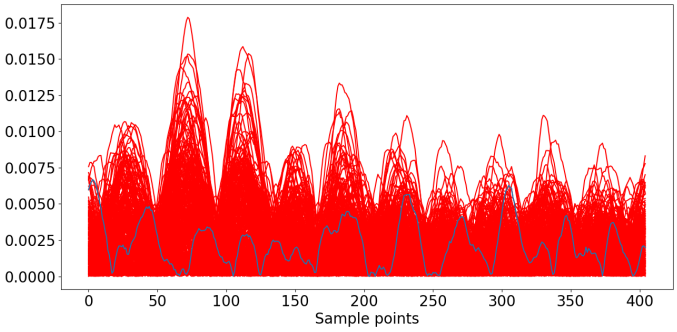


(a) TVLA of unprotected AES C implementation

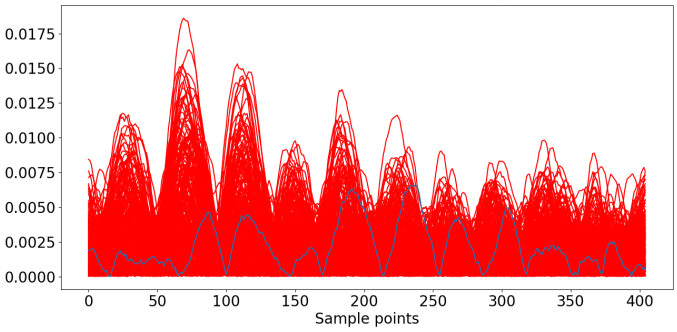


(b) TVLA of masked AES C implementation

Figure 8: TVLA for complete AES implementation



(a) CPA on unprotected AES with ID leakage model

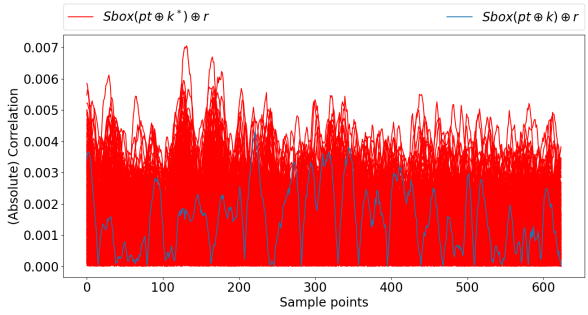


(b) CPA on unprotected AES with HW leakage model

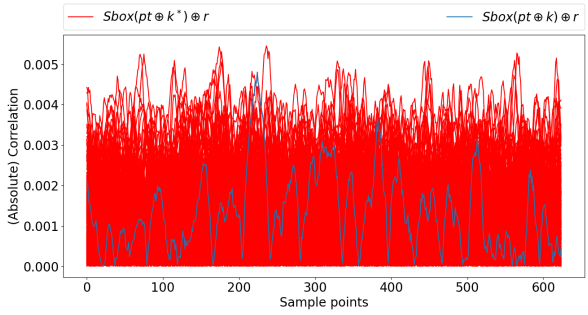
Figure 9: CPA on unprotected implementations

that a higher-order attack without knowledge of the mask will be

even harder to mount. As shown by Figure 10, the correct key does not stand out and hence, it is not recovered.



(a) CPA on masked AES with ID leakage model



(b) CPA on masked AES with HW leakage model

Figure 10: CPA on masked AES with knowledge of mask (567, 582 traces)

As shown by both analyses, we can observe that with CPA even with the knowledge of the implementation and after performing the realignment, the secret key could not be recovered.

Challenges in Profiling and Mitigating Noise from Background Processes. Even though we worked around the unstable trigger via preprocessing, we have not resolved the issue of jittery traces with noise induced by the full OS running in the background. One approach that we hypothesise is if the EM signals corresponding to the background processes and OS can be profiled without AES or S-box operations. Then, it can provide insights into the noise being added during AES implementation. But, we should recall that the chosen processor allows scheduling and supports multi-cores. This means it is impossible to accurately profile the background noise, during the execution of the sensitive algorithms. Additionally, due to the sleep operations (which are essential to managing the issues with the jitter in the trigger), there is more context switching, resulting in noise, that cannot be replicated during profiling.

Therefore, we finally move to deep neural network techniques, that have been shown in the literature, to retrieve secret information, even in the presence of countermeasures, low SNR traces, jitters and de-synchronization.

While more extensive and target-specific pre-processing may result in perfect alignment of the traces allowing CPA-based key recovery. However, there is no standard or proposed method for pre-processing in the literature. Even in real product evaluations, the pre-processing is catered to each target individually. To overcome this factor, in the next section, we explore deep learning based methods which are known to perform pre-processing agnostic evaluations to some extent [11].

4 ANALYSIS USING DL-SCA (PROFILED)

We investigate DL-SCA in a profiled setting. Previous studies have shown that CNNs can perform successful key recovery even in the presence of countermeasures. The shift invariant property of CNN have a natural tendency to overcome countermeasures like jitter or de-synchronization. However, CNN has a large number of hyperparameters to tune (e.g., the number of layers, kernel size, type of activation functions, etc.) compared to machine learning or classical SCA. Furthermore, it has been pointed out that the hyperparameters influence greatly on the performance.

To deal with the hyperparameters, we have performed a basic random parameter search. We run a trial of 100 experiments, in which a CNN is provided with random hyperparameters, and the performance is evaluated on the validation dataset first to determine the best model, and finally tested on a real test set.

4.1 Dataset and Model Training

To prepare the dataset and to find the hyperparameters for training the model, we use the following procedure, which is commonly used in DL-SCA setting:

- We first split the training and test set, and further split the test set into test and validation set.
- We perform a random search on the hyperparameters space to find best model, based on the performance result on the validation dataset.
- After 100 trials, we used the best model obtained to perform the attack on the test set.

We consider 2 scenarios, namely fixed and random key settings for the profiling phase. As the name suggests, in fixed key, the key will be fixed for both the training and testing phase, and in random key setting, the key will be randomized during the training phase and in the attack phase, the key will be fixed. This is to prevent bias during training, where the training/profiling learns about the key instead of the leakage of intermediate values. For experiments conducted under the fixed key setting, we randomized the order of traces and then divided them into training and testing sets.

In Table 2, we describe the different datasets used for the experiments and the number of traces required to perform DL-SCA. It also shows the number of training and testing traces, as well as the number of features used for these different datasets.

4.2 Results for DL-SCA

To evaluate the performance, we use the guessing entropy (GE) metric. We denote $NTGE$ to be the least number of traces required to attain $GE = 0$. In Figure 11, we plotted the GEs obtained for different datasets.

We observed that for unprotected implementation, the secret key can be recovered easily using CNN, as shown by its fast convergence to $GE = 0$, requiring $< 10,000$ traces to recover the correct key byte. We also observed that for masked implementation, we could recover the correct key byte, albeit with more traces. In Table 2, we

Dataset	No of Feats.	No of Traces (%)[2]	No of Train	No of Test	NTGE
Fixed Key	405	303, 692 (10.12%)	100, 000	30, 000	6, 132
Random Key	354	260, 396 (8.14%)	142, 426	30, 000	9, 279
Fixed Key Masked	624	567, 582 (18.92%)	400, 000	100, 000	92, 670
Random Key Masked	904	443, 587 (13.44%)	244, 000	100, 000	75, 647
Full Unprotected AES[3]	2000	2, 889, 890 (72.25%)	200, 000	50, 000	46, 061

Table 2: Dataset used for experiment

present the *NTGE* result for the different datasets. It can be noted that for all datasets, the correct key bytes can be recovered.

The performance of all 100 CNNs while running a random hyperparameter search can be observed in Figure 12 and Figure 13. We can observe that for unprotected AES implementation (both fixed and random keys), with random hyperparameter search, most of the models will eventually converge to $GE = 0$, which means that the DL models have no difficulty in learning the dataset.

On the contrary, for the case of masked implementations, we observed that the performances vary by a lot, and in the end, there is only 1 model for each case, in which the *GE* converges to 0. This indicates that even the masked dataset, DL-SCA is having difficulty in learning the leakage, and by using a random hyperparameter search, one setting eventually leads to a model which can learn the dataset and recover the key.

4.2.1 Results for DL-SCA on Full AES. In previous experiments, the dataset is preprocessed to reduce the time samples. From the CPA results, we know that the pre-processing does not necessarily lead to key recovery, but it helps in the case of DL-SCA. In this case, we consider the case that we do not do feature selection or preprocessing, and use all the traces and the features (2, 000 features).

We plot the results in Figure 14. We observe that in this case, the models eventually have difficulty in learning the leakage, and in the end, only 1 model successfully learns the leakage, in contrast to previously multiple models and on higher *NTGE* as well (46, 061 compared to 9, 279), as shown in Table 2, last row.

5 CONCLUSION

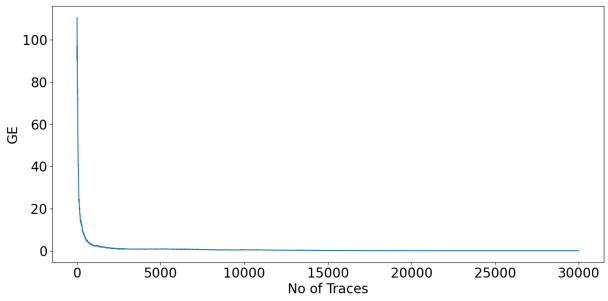
In this work, we investigate the difficulty of mounting a side-channel attack on AES implemented in C on an ARM Cortex A72 processor, with a full OS running in the background. The ARM Cortex A72 operates within a complex ecosystem with an operating system, multitasking, and context switching, resulting in variability and noise. It handles interrupts, daemons, and other background processes that contribute to execution time variability. We confirmed the traces have low SNR and one of the high contributing factors is the misalignment of trace signals incurred by the high jitter. Therefore, we employed preprocessing techniques, to handle some level of de-synchronization, which significantly reduced the number of useful traces. Upon observing TVLA leakage, we proceeded with CPA on filtered traces in known key settings. However, the leakage from the correct key does not stand out.

As there is no standard method to perform pre-processing and is usually adapted for each target, we explore deep learning-assisted side-channel analysis. As proven in existing literature, we can recover the secret key, further highlighting the capability of DL-SCA. Though this is an expected result, we should note that, so far DL-SCA has been used on public datasets that are already vulnerable to classical SCA techniques. But, in this work, we break a dataset with DL-SCA where key recovery with standard CPA is not possible. Even though deep learning (DL) is successful in these attacks, we do not fully understand why it works, necessitating further research on the explainability of deep neural networks. Additionally, appropriate budgeting for countermeasures should be considered, given the evidence that conducting the attack is extremely challenging.

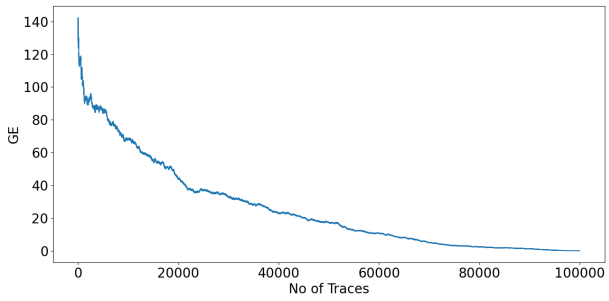
[2]Fixed Key and Random Key datasets were used with a standard AES implementation.
[3]This dataset includes traces from an unprotected AES implementation without any countermeasures.

REFERENCES

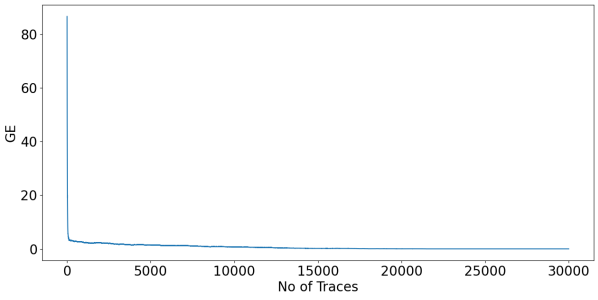
[1] . 2009. AES RD. https://github.com/ikizhvatov/randomdelays-traces
[2] . 2014. DPAv4.2. http://www.dpacontest.org/v4/42_doc.php
[3] 2018. CHES CTF 2018. https://chesctf.riscure.com/2018/news
[4] 2021. ASCADv2. Agence nationale de la scurit des systmes d'information (ANSSI). https://github.com/ANSSI-FR/ASCAD
[5] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. 2003. The EM Side—Channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, Burton S. Kaliski, çetin K. Koç, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 29–45.
[6] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. 2022. DPA, Bitslicing and Masking at 1 GHz. In *Cryptographic Hardware and Embedded Systems – CHES 2015*. Springer-Verlag, Berlin, Heidelberg, 599–619. https://doi.org/10.1007/978-3-662-48324-4_30
[7] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. 2020. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.* 10, 2 (2020), 163–188. https://doi.org/10.1007/s13389-019-00220-8
[8] Donald J. Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, USA, July 1994. Technical Report WS-94-03*, Usama M. Fayyad and Ramasamy Uthurusamy (Eds.). AAAI Press, 359–370.
[9] Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2018. AES HD. https://github.com/AESHD/AES_HD_Dataset
[10] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, Marc Joye and Jean-Jacques Quisquater (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–29.
[11] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, Wieland Fischer and Naofumi Homma (Eds.). Springer International Publishing, Cham, 45–68.
[12] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2003. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, Burton S. Kaliski, çetin K. Koç, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 13–28.
[13] Omar Choudary and Markus G Kuhn. 2014. Efficient template attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12*. Springer, 253–270.
[14] Ibraheem Frieslaar and Barry Irwin. 2017. Recovering AES-128 Encryption Keys from a Raspberry Pi.



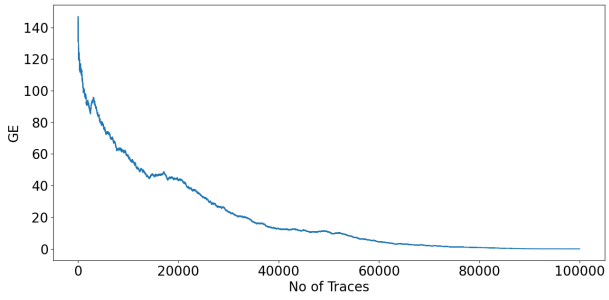
(a) GE on fixed key



(b) GE on masked fixed key

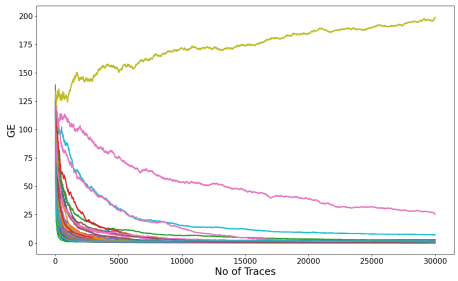


(c) GE on random key

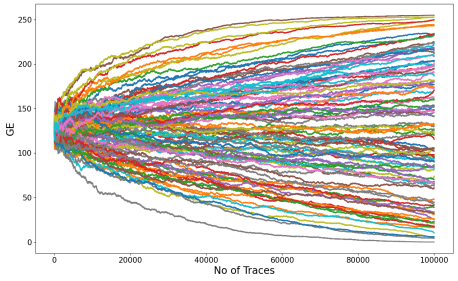


(d) GE on masked random key

Figure 11: Guessing Entropy on DL-SCA

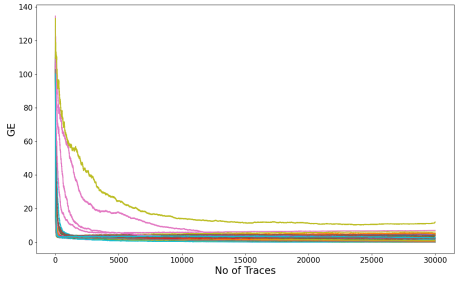


(a) GE on fixed key

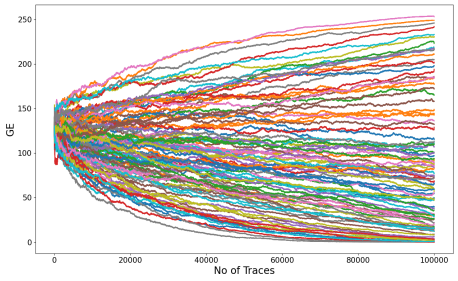


(b) GE on masked fixed key

Figure 12: GE on fixed keys for 100 random searches



(a) GE on random key



(b) GE on masked random key

Figure 13: GE on random keys for 100 random searches

[15] Richard Gilmore, Neil Hanley, and Maire O'Neill. 2015. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 106–111. https://doi.org/10.1109/HST.2015.7140247

[16] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. 2011. A testing methodology for side channel resistance.

[17] Damien Jauvart, Jacques J. A. Fournier, Nadia El Mrabet, and Louis Goubin. 2016. Improving Side-Channel Attacks Against Pairing-Based Cryptography. In *Risks and Security of Internet and Systems - 11th International Conference, CRiSIS 2016,*

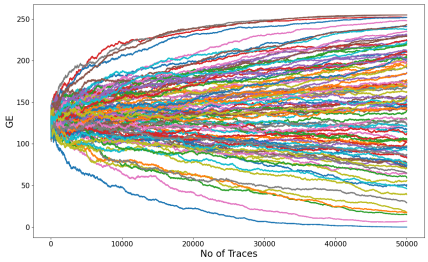


Figure 14: Result on AES Implementation targeting key byte 0 (unprotected C implementation)

Roscoff, France, September 5-7, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10158). Springer, 199–213. https://doi.org/10.1007/978-3-319-54876-0_16

[18] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99)*. Springer-Verlag, Berlin, Heidelberg, 388–397.

[19] H. Li, M. Ninan, B. Wang, and J. M. Emmert. 2024. TinyPower: Side-Channel Attacks with Tiny Neural Networks. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE Computer Society, Los Alamitos, CA, USA, 320–331. https://doi.org/10.1109/HOST55342.2024.10545382

[20] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. Cryptology ePrint Archive, Paper 2016/921. https://eprint.iacr.org/2016/921 https://eprint.iacr.org/2016/921.

[21] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. 3–26. https://doi.org/10.1007/978-3-319-49445-6_1

[22] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. 2019. A Comprehensive Study of Deep Learning for Side-Channel Analysis. *IACR Cryptol. ePrint Arch.* (2019), 439. https://eprint.iacr.org/2019/439

[23] Colin O'Flynn and Zhizhang David Chen. 2014. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*.

[24] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. 2020. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 337–364.

[25] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. 2018. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. Cryptology ePrint Archive, Paper 2018/053. https://doi.org/10.1007/s13389-019-00220-8 https://eprint.iacr.org/2018/053.

[26] Vincent Rijmen and Joan Daemen. 2001. Advanced encryption standard. *Proceedings of federal information processing standards publications, national institute of standards and technology* 19 (2001), 22.

[27] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. 2009. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *Advances in Cryptology - EUROCRYPT 2009*, Antoine Joux (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 443–461.

[28] Benjamin Timon. 2019. Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 2, 107–131. https://doi.org/10.13154/tches.v2019.i2.107-131

[29] Jurian van Geest and Ileana Buhan. 2022. A side-channel based disassembler for the ARM-Cortex M0. Cryptology ePrint Archive, Paper 2022/523. https://eprint.iacr.org/2022/523 https://eprint.iacr.org/2022/523.

[30] Lichao Wu, Guilherme Perin, and Stjepan Picek. 2022. I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. *IEEE Transactions on Emerging Topics in Computing* (2022), 1–12. https://doi.org/10.1109/TETC.2022.3218372

[31] Trevor Yap, Stjepan Picek, and Shivam Bhasin. 2023. Beyond the Last Layer: Deep Feature Loss Functions in Side-channel Analysis. In *Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security*. 73–82.