A Deep Learning-Assisted Template Attack Against Dynamic Frequency Scaling Countermeasures

Davide Galli, *Graduate Student Member, IEEE*, Francesco Lattari, Matteo Matteucci, *Member, IEEE*, and Davide Zoni, *Member, IEEE*

***Abstract*—In the last decades, machine learning techniques have been extensively used in place of classical template attacks to implement profiled side-channel analysis. This manuscript focuses on the application of machine learning to counteract Dynamic Frequency Scaling defenses. While state-of-the-art attacks have shown promising results against desynchronization countermeasures, a robust attack strategy has yet to be realized. Motivated by the simplicity and effectiveness of template attacks for devices lacking desynchronization countermeasures, this work presents a Deep Learning-assisted Template Attack (DLaTA) methodology specifically designed to target highly desynchronized traces through Dynamic Frequency Scaling. A deep learning-based pre-processing step recovers information obscured by desynchronization, followed by a template attack for key extraction. Specifically, we developed a three-stage deep learning pipeline to resynchronize traces to a uniform reference clock frequency. The experimental results on the AES cryptosystem executed on a RISC-V System-on-Chip reported a Guessing Entropy equal to 1 and a Guessing Distance greater than 0.25. Results demonstrate the method's ability to successfully retrieve secret keys even in the presence of high desynchronization. As an additional contribution, we publicly release our `DFS_DESYNCH` database[1] containing the first set of real-world highly desynchronized power traces from the execution of a software AES cryptosystem.**

***Index Terms*—Side-channel analysis, convolutional neural networks, dynamic frequency scaling, deep learning, template attack, RISC-V, AES.**

I. Introduction

OVER the last two decades, Side-Channel Attacks (SCAs) demonstrated a large variety of ways of exfiltrating sensitive information, e.g., the secret key of a cryptographic primitive, from edge and Internet of Things (IoT) computing platforms. SCAs can be divided into *i)* non-profiling, and *ii)* profiling, attacks. Non-profiling attacks, such as Differential Power Analysis [1] and Correlation Power Analysis [2], assume a direct attack on the target device executing a cryptographic algorithm by correlating partially known data being computed, e.g., the plaintexts, and the environmental signal of choice, e.g., the power consumption, generated by the computing platform during the computations. Profiling attacks [3], [4], [5] assume the attacker has access to a copy of the target device to create a profile, e.g., the attack model, later used to attack the target. The more similar the two devices are, the more efficient the attack, as the created profile better matches the target device's behavior. Profiled attacks are generally more powerful and complex to set up than non-profiled ones.

Starting from the introduction of the first profiling SCA, i.e., the Template Attack [3], the field has witnessed a continuous escalation in sophistication over the years. Machine learning [4], [6] and, more recently, deep learning techniques [5], [7], [8] demonstrated the potential to breach into cryptographic devices, thus fueling the continuous investigation of novel countermeasures. Countermeasures can be organized into two classes: masking and hiding. Masking countermeasures [9], [10] split the sensitive intermediate values into different shares that are computed independently to minimize the dependency of each share from the secret key. Hiding countermeasures [11], [12] aim to randomize the target side-channel signal with the goal of reducing the exploitable side-channel information leakage. Notably, the randomization induced by hiding techniques can be defined as adding a correlated noise to the side-channel signal. Indeed, the effectiveness of any hiding technique strongly depends on the difficulty for the attacker to separate the noise from the side-channel information leakage (de-noising). The momentum of hiding techniques is due to the complexity and the cost of designing secure masking solutions compared to the high availability of actuators to implement hiding techniques even on low-end and low-cost IoT devices. The Dynamic Frequency Scaling (DFS) [11], [13] represents a primary cost-effective actuator for power-performance trade-off. The possibility to select the operating frequency in a small set of available operating frequencies at run-time and its wide availability also on edge and IoT platforms [14], [15] make the DFS a suitable candidate to implement hiding countermeasures.

To evaluate and validate the security of desynchronization techniques, such as DFS, having a realistic dataset that reflects the temporal effects introduced by such countermeasures is

Received 11 January 2024; revised 14 September 2024; accepted 29 September 2024. Date of publication 10 October 2024; date of current version 12 December 2024. Recommended for acceptance by X. Jia. *(Corresponding author: Davide Galli.)*

The authors are with Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milan, Italy (e-mail: davide.galli@polimi.it; francesco.lattari@polimi.it; matteo.matteucci@polimi.it; davide.zoni@polimi.it).

Digital Object Identifier 10.1109/TC.2024.3477997

[1]https://github.com/hardware-fab/DLaTA

essential. The lack of such a dataset can severely affect the proposed attacks' practical effectiveness and reproducibility. At present, ASCAD [16] is the first significant effort to provide a reference dataset for the side-channel analysis of temporal desynchronized traces. However, ASCAD provides synchronized data measured from a real-world microcontroller, where temporal desynchronization has been added via data augmentation. For instance, the *shift deformation* and *add-remove deformation* are two data augmentation techniques employed to generate desynchronized traces [17]. Even if data augmentation is a viable technique to face the lack of data, the mathematical function used to augment data must be proved to properly modeled the physical phenomena inducing the desynchronization. Moreover, the open literature highlighted that the complexity of the attack increases with the level of temporal desynchronization up to attack failures above a certain level of temporal desynchronization. To this end, the investigation of highly desynchronized traces using physical actuators is a critical and still open research area.

Contributions - The manuscript presents a novel Deep Learning-assisted Template Attack (DLaTA) against highly desynchronized traces through a commercial Dynamic Frequency Scaling (DFS) implementation, also introducing a new open-source dataset of real-world desynchronized traces to complement the ASCAD database [16]. In particular, the manuscript advances the state of the art in two directions.

- The proposed `DFS_DESYNCH` database and the related computing architecture offer real-world highly desynchronized traces. We leveraged a configurable DFS architecture to introduce randomly temporal desynchronizations in the acquired traces. Current desynchronized datasets highlight a desynchronization between two traces of 100 samples at most [16]. The delivered dataset offers a high temporal trace desynchronization in an average of 64 536 samples (see Section V-B). We also deliver a digital clock labeling architecture to efficiently and precisely monitor the clock frequency variations within each collected trace. Such clock labeling architecture can be employed in place of the high-end oscilloscope used to precisely monitor the computing platform's clock frequency.
- A new Deep Learning-assisted Template Attack (DLaTA) is proposed against desynchronization via DFS. The methodology leverages a profiling attack that employs deep learning to pre-process the highly desynchronized traces, followed by the classic Template Attack to extract the secret key. The pre-processing deep learning pipeline consists of three stages with the final goal of realigning each sample of each trace to a reference clock frequency (f_{ref}). The first and the second stages implement a classifier and a network explanation tool (Grad-CAM), which are used to identify the operating frequency associated with each time windows of each desynchronized trace. Starting from the desynchronized traces and the clock frequency labels identified in the previous stages, the third stage implements an interpolation stage to realign each trace to the reference clock frequency. The experimental results considering a software implementation of the AES cryptosystem executed on an FPGA-implemented RISC-V system-on-chip highlight a Guessing Entropy equal to 1 and a Guessing Distance greater than 0.25, confirming the validity of our proposal.

The manuscript is organized into five sections. Section II discusses the background, while the state of the art is presented in Section III. Section IV details the proposed Deep Learning-assisted Template Attack. The structure of the proposed `DFS_DESYNCH` database is discussed in Section V while the experimental results of our methodology using the `DFS_DESYNCH` are discussed in Section VI with a comparison with two state-of-the-art Convolutional Neural Network. Conclusions and future works are drawn in Section VII.

II. Preliminaries

This section introduces the notation used in this work. Next, it provides primary knowledge about profiled SCAs and deep learning-based SCA, focusing on Convolutional Neural Networks (CNN) and Gradient-weighted Class Activation Mapping (Grad-CAM).

A. Notation

Calligraphic letters like $\mathcal{X}$ are employed to denote sets, and the corresponding upper-case letters (X) indicate their cardinality. The corresponding lower-case letters (**x**, x) denote random variables over $\mathcal{X}$ and their realization, respectively. The symbol $E[\cdot]$ represents the expected value and might be subscripted by sets or random variables $E_{\mathcal{X}}[\cdot]$ to specify under which probability space it is computed. A dataset $\mathcal{T}$ is a collection of side-channel measurements (traces). Each trace t_i is associated with an input value (plaintext) p_i and a key k_i. Here, k denotes a key candidate taking its value from the key space $\mathcal{K}$, and k^* represents the correct key. In the deep learning domain, $\mathcal{F}$ denotes the set of labels with a label f_i for each frequency class i, and **y** indicates the network's output, with an element y_f for each class.

B. Template Attacks

In profiled SCAs, it is assumed that the attacker has access and complete control of a copy of the device they want to attack. This way, the attacker can collect as many measurements as needed, and they can build a profiling model of the leakage. During the attack, a few traces collected from the attacked device are generally sufficient to infer the secret key, thanks to the profiling model. The best-known profiled attack is the Template Attack [3], where Bayes' theorem is used to obtain predictions. Commonly, Template Attacks are based on multivariate normal distribution characterized by the mean and covariance matrixes. During the profiling phase, these two matrixes are built having an entry for each $(p, k) \in \mathcal{P} \times \mathcal{K}$ and used in the attack phase to match the template with unseen leakage traces. Before profiling, the measurements are often pre-preprocess to help the template to infer the secret key, and the measurements' dimensionality is reduced using approaches like Principal Component Analysis (PCA) [18] or Linear Discriminant Analysis (LDA), focusing on the most leaking Points of Interest.

C. Profiled Deep Learning-Based SCA

In supervised machine learning algorithms, the aim is to learn a function $g\colon \mathcal{X} \rightarrow \mathcal{Y}$ that maps an input to the output based on examples of input-output pairs. The function g is parametrized by θ^z, where z represents the number of trainable parameters and θ denotes the vector of parameters (also called weights) learned in a profiling model. In the side-channel domain, the input is a 1D time series, and each sample corresponds to a feature. During the profiling phase, the goal is to learn the vector θ by minimizing the empirical risk represented by an ad-hoc loss function using a training dataset. In the attack phase of the proposed approach, the aim is to predict the probabilities of frequency class label f_i of an unseen dataset using the trained model g. Indeed, the function g is realized through a deep neural network with the Softmax output layer for multi-class classification.

Convolutional Neural Networks - Convolutional Neural Networks (CNNs) are a well-known deep learning architecture, and they mainly consist of three types of layers: convolutional layers, pooling layers, and fully connected layers. Convolutional layers are characterized by a set of filters (or kernels). The size of the filters is usually smaller than the actual input data. During the convolution, the filter slides across the sample data, and the dot product between every element of the filter and the input is calculated at every spatial position. Pooling layers reduce the data dimensions by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Fully connected layers connect every neuron in one layer to every neuron in another layer and are associated with an activation function. These three layers are often augmented by a fourth one, i.e., the batch normalization layer, which makes training more stable by mitigating the phenomenon of the internal covariate shift.

Gradient-weighted Class Activation Mapping - Gradient-weighted Class Activation Mapping (Grad-CAM) [19] is a technique to understand how CNNs make decisions. Grad-CAM creates a heatmap highlighting the input regions most important to CNN's classification. This heatmap can then facilitate input segmentation, which, in the context of this work, involves assigning a frequency label to each sample based on its temporal window. A significant benefit of employing Grad-CAM is its ability to operate without the need for additional training. It leverages a pre-trained classification network, thereby streamlining the segmentation process. Typically, segmentation tasks demand more complex networks and extensive training. However, Grad-CAM circumvents this by reducing the training requirement to a classification problem. Moreover, Grad-CAM enables the creation of a segmentation model even when feature- wise labels for training are absent. This is particularly advantageous since such labels are prerequisites for specialized segmentation task architectures.

Grad-CAM has been demonstrated to provide high-resolution and class-discriminative visualizations when applied to image classification tasks [19]. Inspired by the object localization performance obtained by Grad-CAM in the image domain, we built upon the same ideas to localize the regions referring to the different frequencies in the power traces, which, for the considered task, we demonstrate to be a valid approximation of the desired segmentation.

III. Related Works

The introduction by Chari et al. in 2002 of the Template Attacks [3] sparked the discussion about profiling SCAs. Template Attacks have been shown to be information-powerful attacks, but they are also based on assumptions such as the noise follows a Gaussian distribution, and the attacker can collect as many traces as they want for profiling [20], [21]. Despite these limitations, Template Attacks demonstrate in practice good performance and ease of deployment due to their few hyper-parameters requiring fine-tuning. Generally, before profiling, a feature engineering phase is required in which a subgroup of Points of Interest (POI) is selected. This feature selection can use a machine learning approach [22] or dimensionality reduction techniques like Principal Component nalysis (PCA) [18], Linear Discriminant Analysis (LDA) [23], or Sum of Squared Pairwise T-differences (SOST) [24].

More recently, deep learning-based SCAs have become highly appreciated, especially in cases where assumptions about data distributions cannot be made or SCA countermeasures are implemented [5], [25], [26]. Benadjila et al. test in [16] various neural network architectures by exploring their hyperparameters and concluded that CNN models should be preferred in the context of SCA. Their best CNN model consists of five convolutional blocks followed by two final dense layers, each comprising a convolutional layer, ReLU activation function, and an average pooling layer. The number of convolutional filters ranges from 64 to 512, with a kernel size of 11. However, in the presence of high desynchronization, even their best CNN cannot achieve Guessing Entropy 1. Hettwer et al. suggest using Layer-wise Relevance Propagation (LRP) to perform SCA [27]. LRP is a technique that comes from CNN literature, showing which parts of the input data are most important for CNN's output. The attacker can find the correct key by comparing these parts with different key guesses.

Although deep learning-based SCAs are very successful, nowadays, the Template Attack still has certain advantages. The most significant one is that it has only a few hyperparameters to tune, making it easier to use. Starting from this statement, Wu et al. [7] used a hybrid approach where deep learning is used as pre-processing to find an efficient embedded representation of input data. The actual attack is then performed by learning a template on such representation.

Desynchronization techniques, such as DFS, serve as a natural countermeasure against various SCAs that rely on synchronous measurements. One common approach involves dynamically changing the operating frequency of the computing platform at runtime using a DFS actuator. The first work using this technique is reported by Yang et al. [11], which explored the idea of Random Dynamic Voltage and Frequency Scaling (RDVFS) to undermine the effectiveness of SCAs by desynchronizing in time and values the collected power traces. However, the discrete number of frequency switching in the

proposed countermeasure reduces the efficiency of the desynchronization in [11]. To improve on Yang's design, other solutions use random DFS actuators for Xilinx FPGAs [13], [28], showing how to prevent Correlation Power Analysis (CPA) attacks with 4 million traces. Hettwer et al. [13] exploit the reconfigurability of Xilinx MMCMs to change the frequency configuration after a certain number of AES encryptions. Their design uses a single MMCM with up to eight clock outputs delivering up to 2 056 frequencies, while a clock multiplexer randomly switches the output clock when the cryptographic core is running. The authors show how their design can resist powerful CNN attacks based on [27], by changing the MMCM configuration every 10^3 encryptions. Dao et al. [28] implement a random DFS countermeasure for FPGA that changes clock frequency every AES computation. They show that with 219 000 different frequencies, this method makes CPA fail with 5 million traces. However, their DFS is vulnerable to a deep learning attack using the CNN proposed by [16], which recovers 13 out of 16 bytes. Notably, commercial DFS implementations [14], [15] typically employ a limited number of frequencies, as opposed to the thousands in academic implementations. These commercially available DFS solutions prioritize practicality and efficiency while providing significant desynchronization against SCAs.

Two strategies are usually used to circumvent desynchronization techniques such as clock jitter [17], Random Delay Insertion [12], and shifting [16]. The first consists of increasing the number of traces used for profiling by a factor proportional to the desynchronization magnitude. Conversely, the second is to apply various realignment techniques to resynchronize the multiple traces. Durvaux et al. [29] proposed the use of Hidden Markow Models pattern recognition to remove any dummy operation used as random delay. Other techniques are signal-processing oriented and more adapted to hardware countermeasures [30], [31]. A recent approach based on autoencoders has been presented by We and Picek in [32]. Their proposed neural networks can clean up traces and re-aligning them by treating desynchronization as noise. However, the training of these autoencoders presupposes the availability of a device that can selectively deactivate countermeasures. Nevertheless, none of these methodologies tackle the desynchronization introduced by a real DFS. Furthermore, the utilization of ASCAD complemented by data augmentation raises some objections. The ASCAD dataset, introduced by Benadjila et al. in [16], helped the research community to find increasingly deep learning-based attacks that could circumvent the most commonly used countermeasures, such as masking and desynchronization [32], [33], [34]. The ASCAD dataset represents a common target for profiling SCA as it contains measurements protected with a masking countermeasure and settings with fixed or random keys. Nevertheless, the desynchronization provided by ASCAD is artificially made in post-processing and considers one fixed frequency, which does not consider the edge effects that occur during a physical frequency change due to a DFS actuator. For these reasons, attacks and algorithms assessed on the ASCAD dataset [32], [34] could show different behaviors in real-world scenarios.

IV. Methodology

This section presents a novel Deep Learning-assisted Template Attack (DLaTA) methodology targeting physically desynchronized traces. Starting from the highly desynchronized traces, the attack methodology implements a pre-processing deep learning stage to resynchronize the traces to a reference clock frequency before applying a classical template attack. Notably, the proposed attack methodology exploits trained stages, i.e., the deep learning and the template stages, thus, a more natural presentation flow requires organizing the discussion into training and attack phases as depicted in Fig. 1.

Training Phase - The phase is meant to train the templates and a CNN frequency classifier starting from an input dataset of pairs. Each pair consists of a highly desynchronized power trace and the corresponding set of temporally ordered labels identifying the clock frequency of each interval in the trace. The templates are trained using a two-stage pipeline. First, the `Trace Interpolation` stage resynchronizes all the traces of the input dataset to a common reference clock frequency. Notably, the `Trace Interpolation` stage leverages the clock frequency labels associated with each sample of the input traces to perform the resynchronization. Second, the templates are trained using the resynchronized traces. A minor aggregation of N_{agg} samples over time is performed by the `Aggregation` stage to mitigate interpolation errors.

The CNN is trained using a two-stage pipeline with the final goal of predicting the clock frequency of a given input trace. To this end, we employed a `CNN Dataset Creation` stage to create the training dataset by selecting a set of time windows from each trace which exhibit a constant clock frequency (see $Win_1, ..., Win_N$ in `Training Phase` of Fig. 1). The availability of a database with clock frequency labels for each trace sample makes this `CNN Dataset Creation` procedure a trivial task. Finally, starting from the extracted constant clock frequency time windows, the `CNN Training` stage delivers the `Trained CNN`.

Attack Phase - The attack phase implements the DLaTA, which consists of a pre-processing deep learning pipeline (see `Segmentation` and `Trace Interpolation` stage in Fig. 1) to resynchronize the traces to a reference clock frequency (f_{ref}) and the classical template attack (see `Template Attack` stage in Fig. 1). The pre-processing deep learning pipeline consists of three stages. The first and second stages implement a frequency classifier and a network explanation tool used to identify the operating frequency associated with each time window of a desynchronized trace (see `Segmentation` stage in Fig. 1). Notably, the frequency classifier has been trained using constant clock frequency traces, while in the Attack Phase, it is fed with highly desynchronized traces. Thus, the output vector is expected to highlight various clock frequencies associated with each desynchronized trace. The main novelty of the proposed approach is to avoid the direct use of such output vector. In contrast, we feed the output vector obtained from the frequency classifier into the Grad-CAM CNN explanation tool. Grad-CAM highlights the portions of a desynchronized trace that activated a specific frequency class in

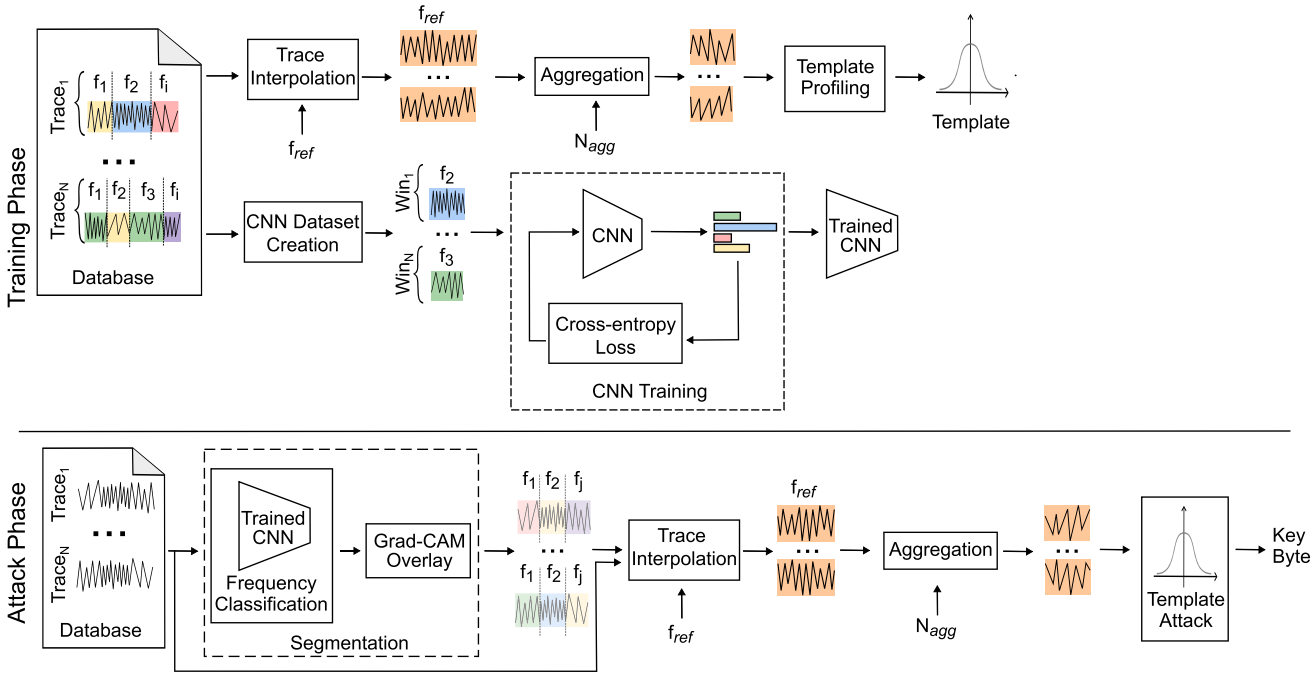


Fig. 1. Overview of the proposed deep learning-assisted template attack methodology for physically desynchronized traces highlighting the training and attack phases.

the output vector, thus allowing to associate a clock frequency label to each sample of the desynchronized traces. Starting from the desynchronized traces and the clock frequency labels identified in the `Segmentation` stage, the third stage of the deep learning pipeline applies the `Trace Interpolation` stage to realign each trace to the reference clock frequency before performing the classical template attack. An N_{agg}-samples aggregation over time is used to fix minor misalignments due to the rough frequency segmentation and to mitigate interpolation errors (see `Aggregation` stage in Fig. 1).

The rest of this section delivers a detailed discussion targeting the dataset creation, the trace interpolation, the CNN for frequency classification, and the segmentation, i.e., the key elements of the proposed solution, in Sections IV-A–IV-D, respectively.

A. CNN Dataset Creation

Considering the training pipeline, the `CNN Dataset Creation` stage takes the side-channel traces in input and generates the dataset to train, test, and validate the CNN (see `Training Phase` in Fig. 1). The CNN dataset is built starting from the available collection of side-channel measurements, consisting of power traces desynchronized through DFS and a frequency label for each sample (see Section V). Thus, each initial power trace encloses multiple frequencies, with possible values from a predefined finite set $\mathcal{F}$. The neural network is trained to classify each power trace according to its clock frequencies, which necessitates training data with constant frequency. The `CNN Dataset Creation` procedure inputs a desynchronized side-channel trace and selects N-sample windows of constant frequency. Since the starting database provides the frequency value for each sample, selecting a constant frequency window is trivial. Padding is left before and after each frequency change to avoid border effects due to the physical actuator. The output CNN dataset consists of N-sample constant-frequency windows with relative frequency labels (see $Win_1, ..., Win_N$ in `Training Phase` of Fig. 1).

B. Trace Interpolation

The `Trace Interpolation` stage is used both during the training phase and the attack phase to resynchronize the traces to a reference clock frequency (f_{ref}). The procedure employed to interpolate the trace is reported in Algorithm 1. The algorithm takes as input the desynchronized trace *t*, the segmentation *s*, and the desired reference frequency f_{ref}. The input segmentation *s* consists of a vector of frequency values for each sample in *t*. Notably, during the training phase, *s* is known and provided directly as part of the available database. Instead, during the attack phase, *s* is estimated by the proposed segmentation method (see `Segmentation` stage in Fig. 1). Given the input segmentation, all the intervals characterized by a single clock frequency, named *windows* in the algorithm, are extracted from the original trace (see line 2 in Algorithm 1). The algorithm considers then one window at a time and interpolates based on the window frequency (f_{win}). If f_{win} is the same as the reference frequency, then the interpolation is unnecessary since the window is already aligned with the reference frequency (see lines 7-9 in Algorithm 1). In that case, the samples within the original window are directly stored in the output array t_{int}. On the contrary, in case f_{win} and f_{ref} are not the same, a linear interpolation is performed, represented by the *scaleWindow*

Algorithm 1 Trace Interpolation Algorithm

Input t, s, f_{ref}
{t: Input trace}
{s: Frequency segmentation for input trace t}
{f_{ref}: Reference frequency}
Output t_{int}
{t_{int}: Interpolated trace}

1: **procedure** TRACEINTERPOLATION(t, s, f_{ref})
2: $windows \leftarrow$ getCostantFrequencyWindows(t, s)
3: $t_{int} \leftarrow$ [ ]
4: **for** win in $windows$ **do**
5: $f_{\text{win}} \leftarrow$ getFrequency(win)
6: **if** $f_{\text{win}} = f_{\text{ref}}$ **then**
7: // No interpolation required
8: $t_{int} \leftarrow$ concat(t_{int}, win)
9: **else**
10: // Interpolation required
11: $win_{int} \leftarrow$ scaleWindow($win, f_{ref}/f_{win}$)
12: $t_{int} \leftarrow$ concat(t_{int}, win_{int}])
13: **end if**
14: **end for**
15: **return** t_{int}
16: **end procedure**
17: **procedure** SCALEWINDOW($window$, $freqRatio$)
18: $samples \leftarrow$ len($window$)
19: $numSample_{int} \leftarrow samples/freqRatio$
20: // Create new samples' list from 0 to $numSample_{int}$
21: $samples_{int} \leftarrow$ newList($0, numSample_{int}$)
22: // Linear interpolation from *samples* to $samples_{int}$
23: $window_{int} \leftarrow$ interp1D($windows, sample,$ $sample_{int}$)
24: **return** $window_{int}$
25: **end procedure**

function (see lines 10-13 and 17-25 in Algorithm 1). The interpolation scales the considered window by a factor $f_{\text{ref}}/f_{\text{win}}$, thus increasing or decreasing the number of samples in the final trace. The obtained scaled window is then stored in the output array t_{int}. The procedure is repeated for each window until all the constant-frequency windows in t have been processed. Finally, the interpolated trace t_{int} is returned (see line 15 in Algorithm 1).

C. Convolutional Neural Network

Fig. 2 shows the architecture of the 1D CNN for classification, which is adapted from a 2D ResNet [35] to use the residual learning framework for the analysis of 1D sequential data [36]. A 1D ResNet architecture has been choosen for its effectiveness in processing one-dimensional data like side-channel traces, capturing local and global dependencies. The ResNet structure facilitates efficient gradient propagation, improving the training procedure, which is crucial for extracting subtle patterns from 1D time-series. Our network aims to classify the operating frequency of the input side-channel trace. The input to the CNN is a power trace (t) of N samples, while its output is a classification vector (y) with a probability for each frequency in the label set $\mathcal{F}$. The CNN architecture comprises six pipelined blocks: a convolutional block, two residual blocks, a global average pooling layer, two fully-connected layers, and a softmax layer. Each convolutional block includes a 1D convolutional layer, a batch normalization layer, and a ReLU activation function. The residual block [35] implements two convolutional blocks enhanced with shortcut connections to sum the features element-wise (see Fig. 2). Zero-padding is performed when the number of filters between the input and output of the shortcut connection is different. All the 1D convolutional layers implement a kernel with size 20, a stride equal to 1, and zero padding to keep the number of samples equal to N. The first convolutional layer and the one in the first residual block implement 16 filters, i.e., the shape of the output tensors is $N \times 16$. The second residual block increments the number of filters to 32, thus, the shape of the output tensor after the two residual blocks is $N \times 32$. The global average pooling layer averages the obtained features over the temporal dimension N, thus delivering a feature vector of size 1×32. The feature vector is then fed to the first fully-connected layer with a ReLU activation function that reduces the input dimension to an encoded representation of 24 values. The second fully-connected layer outputs a linear vector with F dimensions, which is equal to the number of frequencies in $\mathcal{F}$ (see Section V for more details). Finally, the softmax layer outcomes a classification vector with the probability values for each frequency. Notably, the structure of the global average pooling layer allows the eventual use of different N values for the training and attack pipelines.

CNN training - The designed CNN is trained using constant clock frequency windows with N samples extracted from the available database (see Section IV-A). The network parameters θ are obtained through an iterative optimization procedure. At each step, a trace window t is fed into the network, and it produces an output denoted by $y = g(t)$, where g is the non-linear function realized by the network. The y output represents the network's classification of the input window, expressed as probabilities across the F possible frequencies. To guide the training process, the CNN's output probabilities are compared with the actual frequency lables f of the input trace. Equation 1 misures this comparison as cross-entropy loss.

$$\mathcal{L}(y, f) = -\sum_{j=1}^{F} f_j \log y_j = -\sum_{j=1}^{F} f_j \log g_j(t, \theta) \qquad (1)$$

The loss value is then used to adjust the network's parameters θ, progressively improving its ability to distinguish between different frequency patterns in the data.

D. Segmentation

Considering the attack pipeline, the trace segmentation block (see `Segmentation` stage in Fig. 1) assigns a frequency value to each sample in the attacked desynchronized traces. Given an input trace t with N samples in the trace, the segmentation is the 1D vector s of N elements with values $s_i \in \{f_1, f_2, \ldots, f_F\}$, where f denotes a frequency label and F the number of possible

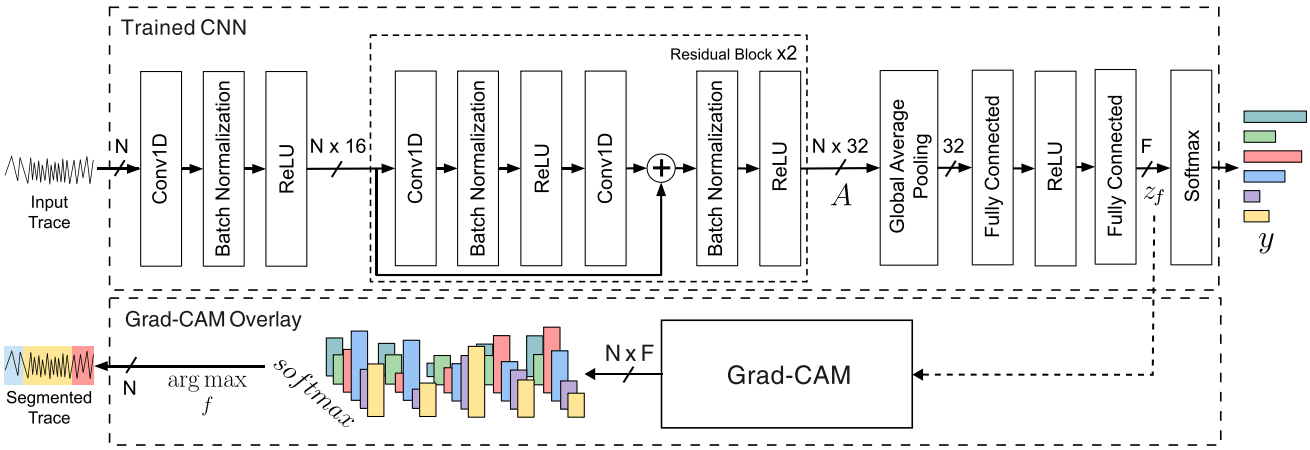


Fig. 2. Architecture of the frequency classification network and Grad-CAM based segmentation.

frequencies synthesized by the DFS. We exploit the Grad-CAM localization capability to approximate the segmentation s.

Grad-CAM based trace segmentation - The Grad-CAM method consists of a sequence of steps, from the CNN's classification output to the trace segmentation (see the bottom of Fig. 2). First, the CNN's classification output z is examined before the final Softmax layer. This output z is a vector containing scores for each of the F possible frequencies within the trace. Each element z_f reflects the network's confidence that the input trace corresponds to a specific frequency.

Next, a localization map *Grad-CAM* is generated for every frequency. These maps indicate the importance of each sample within the trace for predicting each frequency. The localization maps are constructed by combining the feature maps, denoted by A, extracted from the final residual block of the CNN (see layer A in Fig. 2).

Finally, the localization maps segment the trace by assigning frequencies to individual samples. The segmentation is achieved by selecting the frequency with the highest localization score for each sample after applying a normalization step using the Softmax function. The output s indicates the frequency assigned to each sample i in the trace. Equation 2 denotes the segmentation s_i for each sample i.

$$s_i = \arg\max_f \text{Grad-CAM}_i^f \tag{2}$$

V. The DFS_DESYNCH Database

Taking steps from the ASCAD database [16], we publicly open the measurements used for our experiments. The ASCAD database collects EM-based side-channel measurements from different protected and unprotected software-implemented AES-128. Over the years, ASCAD has become the reference standard for benchmarking machine and deep learning attacks by the SCA community. Considering the desynchronized trace scenario, ASCAD offers Python scripts allowing the users to desynchronize the traces rigidly, i.e., effecting a random translation within a user-defined range. Our `DFS_DESYNCH` database consists of highly desynchronized power traces of a software-implemented AES-128 [37] executed on an FPGA-implemented RISC-V computing platform. The proposed architecture leverages the DFS actuator to randomly scale the operating clock frequency for desynchronization purposes. By publishing our database, we aim to complement ASCAD by offering real-world, highly desynchronized power traces to the broader research community. While ASCAD employs a mathematical model for desynchronization, our traces leverage an actual DFS process. This approach provides a more realistic desynchronization scenario, where the desynchronization is not constant throughout the trace but varies with each frequency change. Additionally, the DFS mechanism introduces real noise affecting all traces, with the transition between frequencies further amplifying this noise. The database is available at https://github.com/hardware-fab/DLaTA.

A. Acquisition Setup

Fig. 3(a) depicts the acquisition setup consisting of a host computer, two Piscoscope 5244d digital sampling oscilloscopes (DSO), i.e., OS_{pwr} and OS_{clk}, and a NewAE CW305 board, which features a shunt resistor in the Vdd path of the target AMD Xilinx Artix-7 FPGA. Using a shunt resistor and decoupled power rails allows measuring a single quantity, i.e., voltage drop, rather than voltage and current.

The host communicates with the FPGA, sending inputs to the computing platform and receiving its results. The communication interface between the host and SoC leverages a 1.5 Mbps UART. The DFS module is developed following the design in [38]. It receives as input the new frequency $\texttt{f}_i$ to set. The input is provided by a True Random Number Generation (TRNG) [39] so that the DFS clock output $\texttt{clk}_o$ always has a random period. The employed TRNG is a 3-bit NLFIRO with LFSR post-processing [39], its randomeness has been evaluated using the NIST SP 800-22 test suite [40] with all tests passing successfully. The DFS module clocks only the computing platform, while a constant reference clock drives the rest of the SoC. The computing platform takes as inputs `plaintext` and `key` sent by the host and outputs `ciphertext`, i.e., the result of the AES encryption, and a `trigger` signal. The `trigger` signal

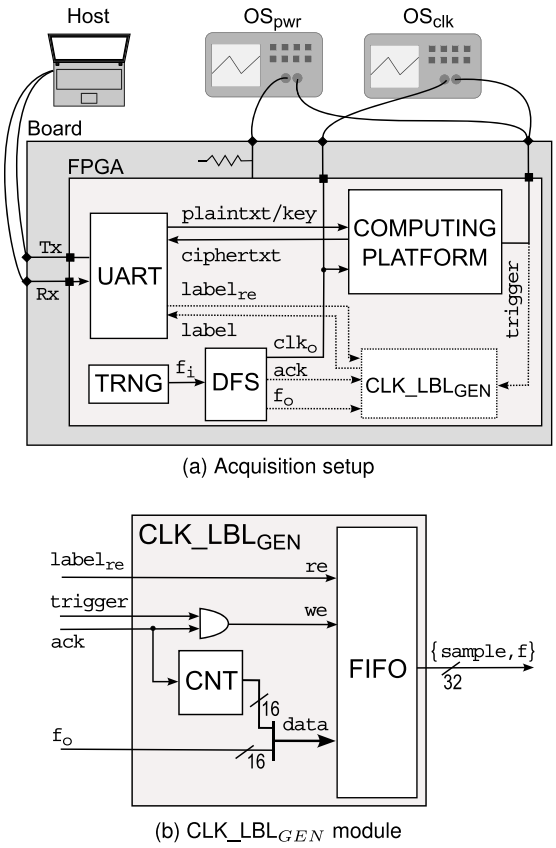


Fig. 3. Block diagram of acquisition setup and CLK_LBL$_{GEN}$ module for generating digital labels.

starts the synchronous data acquisition of the two oscilloscopes when the AES encryption begins. In particular, OS_{pwr} measures the power consumption of the computing board, while OS_{clk} is connected to the DFS clock output $\texttt{clk}_o$. The `ciphertext` from the computing platform is sent back to the host, which checks the correctness of the encryption.

Digital clock labeling - From the theoretical standpoint, the clock frequency variations can be back-annotated on the collected traces by calculating the length of each period of the clock signal measured by the OS_{clk} oscilloscope. Despite its theoretical simplicity, such a task requires recording a reasonably good square wave of the clock signal, thus imposing a high-performance oscilloscope with a sampling rate at least 20 times faster than the clock frequency generated by the DFS module. To this end, we propose a digital labeling module to avoid using a high-performance OS_{clk} oscilloscope (see CLK_LBL$_{GEN}$ in Fig. 3(b)).

The CLK_LBL$_{GEN}$ consists of a free-running counter and a storage element. The free-running counter counts the number of elapsed clock cycles at the reference frequency elapsed from the beginning of the computation. At each clock frequency variation, the CLK_LBL$_{GEN}$ stores a 32-bit value where 16 bits are the current value of the free-running counter and 16 bits represent the hardware configuration fed into the DFS module to synthesize the new clock frequency. Such values are periodically downloaded to the host computer via the SoC UART. Notably, the operating frequency can be uniquely identified for each 32-bit record from the 16 least significant bits. To this end, each 32-bit record reports the newly applied operating frequency and the time, in terms of number of elapsed clock cycles at reference clock frequency, at which the frequency change happened.

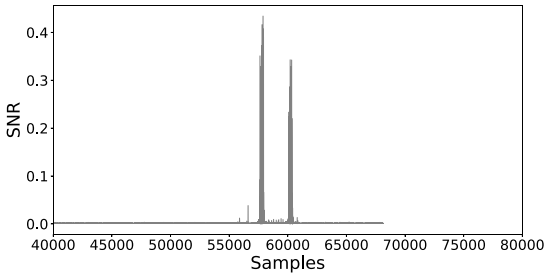
B. Database Structure

`DFS_DESYNCH` database records power traces of a software AES implementation executed on an in-house RISC-V System on Chip [41], [42], [43], used as the reference computing platform. As cipher implementation, we selected the constant-time, unprotected version of AES-128 from the OpenSSL software codebase [37]. The DFS has been configured to synthesize six clock frequencies between 35 MHz and 60 MHz with a step of 5 MHz. In particular, the DFS is set to continuously synthesize a new frequency randomly selected between the available ones, resulting in up to 50 clock frequency changes for a single trace. The entire `DFS_DESYNCH` dataset consists of 256 000 traces, 1 000 for each possible key byte, composed of 200 000 time samples. Each trace has been sampled at 125 Msample/s with a 12-bit resolution. Each power trace records the whole AES encryption, regardless of the operating frequency. Notably, traces with an average faster frequency have the last part of the acquisition in idle, as they have already finished computation. Moreover, several hundred random operations were performed before each encryption to further desynchronize the side-channel traces. The collected power traces have been high-pass filtered with a cut-off frequency of 125 kHz to remove any low-frequency signal component added by the physical DFS actuator.

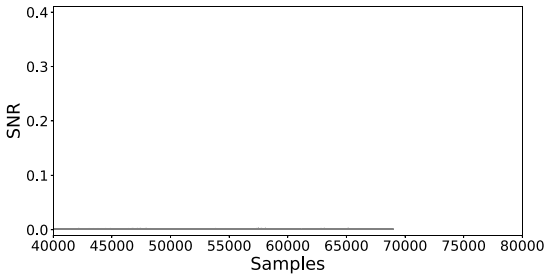
To identify the leakage samples related to the first sbox computation during the first round, namely $\texttt{sbox(p[0]} \oplus \texttt{k[0])}$, the Signal-to-Noise Ratio (SNR) has been processed with a total of 100 000 side-channel traces. Notably, the transition of the operating frequency is random and can also happen precisely at the execution of the `sbox` operation. Moreover, even if the `sbox` operation and the surrounding samples are collected at the same frequency, such frequency can be different for each trace. Fig. 4 plots the SNR for unprotected traces, i.e., with a static frequency of 45 MHz, and for traces in the `DFS_DESYNCH` database. It is clearly visible how DFS greatly reduces leakage; indeed, side-channel traces from `DFS_DESYNCH` database have an SNR without any peak and a maximum value 94 times less than the static traces.

Fig. 5 depicts the `DFS_DESYNCH` database architecture. The Hierarchical Data Format version 5 (HDF5) was chosen to save the dataset. The `dfs_desynch.h5` file has the following structure:

- Two main groups, `profiling` and `attack`, separate the traces collected for the profiling and attack phase, containing each N_{SW} traces. We collected two batches of N_{SW} traces on two separate days and randomly split them between the profiling and attack group. N_{SW} is set to 128 000, namely 500 traces for each possible key byte.



(a) Static side-channel traces at 45 MHz.



(b) DFS_DESYNCH side-channel traces.

Fig. 4. SNR related the intermediate sbox(p[0] ⊕ k[0]) in the interval [40 000, 80 000] including the first AES round.

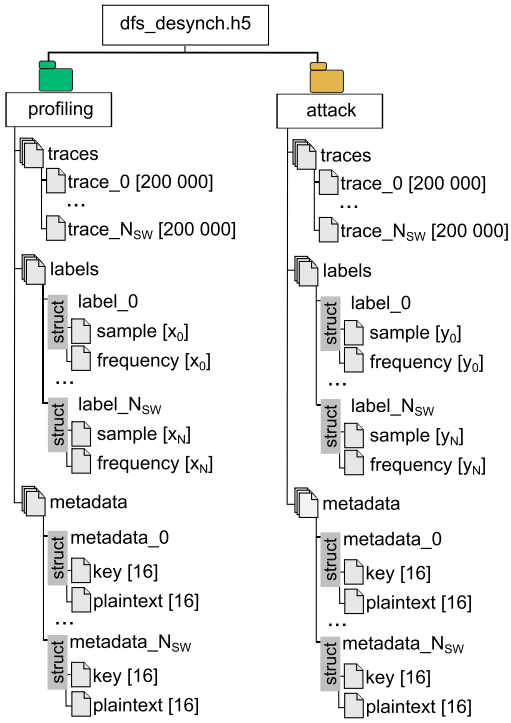


Fig. 5. Structure of the dfs_desynch.h5 HDF5 data files.

- Each group includes three datasets within it:
  - The traces dataset contains N_{SW} power traces, each made of 200 000 time samples recording the whole AES encryption preceded by hundreds of random instructions. The power traces are post-processed using a highpass filter with a cut-off frequency of 125 kHz. The traces have a high desynchronization rate, with an average rate of 64 536 samples and a standard deviation of 15 053.17. The maximum deviation between the fastest and slowest trace is 140 833 samples.
  - The labels dataset contains N_{SW} labels for each power trace, indicating the frequency changes. The members of the compound dataset are two arrays of unsigned 16-bit integers. *(i) Sample* marks the time sample when occurs a frequency change and ranges from 0 to 200 000; *(ii) Frequency* specifies the new operating frequency starting from the corresponding sample, and it takes value from the set $\{35, 40, 45, 50, 55, 60\}$. The first sample is always 0, providing the starting frequency. The member's array dimensions, x_i and y_i, are not fixed since a random number of frequency changes can occur for each trace. On average, each trace has 41 frequency changes with a variance of 2.60.
  - The metadata dataset contains one metadata for each trace, including *(i) key* and *(ii) plaintext*. Only the first key byte changes every 500 traces. Intermediate values to perform the attack can be derived using the metadata, such as sbox(p[0] ⊕ k[0]).

We also public a Python script to generate the dataset used to train our CNN, as described in Section IV-A.

VI. Experimental Evaluation

This section discusses the experimental evaluation of the proposed DLaTA method. Section VI-A details the experiments and evaluations of the interpolation procedure. Section VI-B reports the dataset and CNN training parameters and results. Section VI-C evaluates the experiments of the Grad-CAM-based segmentation. Section VI-D assesses the Template Attack upon the restored traces. Finally, Section VI-E compares our approach with two state-of-the-art CNNs designed for SCAs.

A. Trace Interpolation

The interpolation block is designed to recover information lost due to trace desynchronization. The idea is to evaluate the quality of the information recovered from the desynchronized traces by computing the similarity between interpolated and static traces.

Quality metrics - We opted for the Pearson Correlation Coefficient (PCC) to quantify the similarity between the interpolated and static traces. Given two traces t_a and t_b, PCC is defined by Equation 3, where $Cov(\cdot,\cdot)$ is the covariance between two variables and $Var(\cdot)$ is the variance.

$$PCC(t_a, t_b) = \frac{Cov(t_a, t_b)}{\sqrt{Var(t_a)Var(t_b)}} \tag{3}$$

The coefficient value ranges from -1 up to $+1$, and an absolute value of 1 denotes a perfect linear relationship.

Configuration - We compared subsets of traces from two sources: power traces from the DFS_DESYNCH database and a separate set of static traces acquired with no desynchronization enabled. Both subsets included 500 traces with the same key

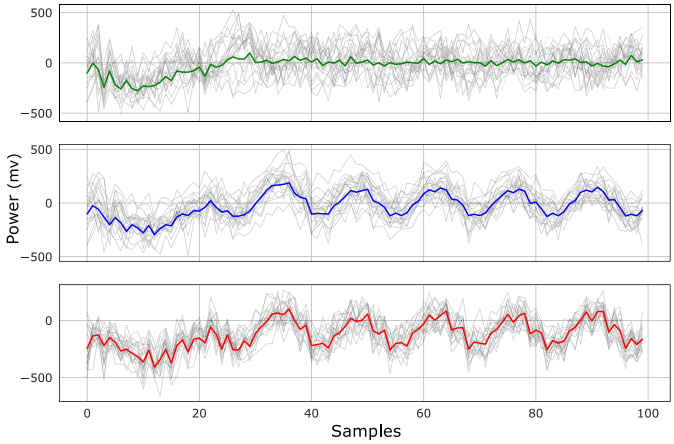


Fig. 6. Qualitative comparison between the DFS_DESYNCH database (first row), the corresponding interpolated traces (second row), and power traces acquired with no DFS enabled (third row), over a range of 100 samples. Each colored line is the sample-wise average.

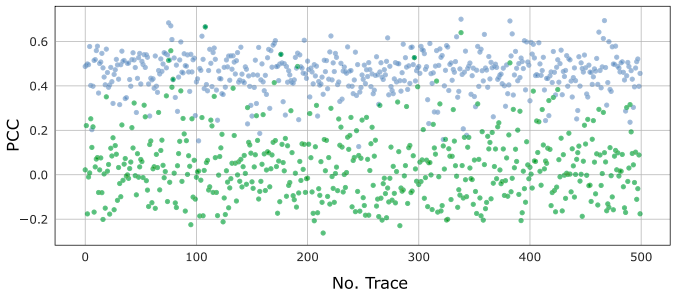


Fig. 7. Pearson Correlation Coefficient computed between static-desynchronized traces (green points) and static-interpolated traces (blue points).

value, and their reference frequency was set to 45 MHz. For this comparison, we used the true labels of the frequencies instead of the learned segmentation model to focus only on the interpolation performance. For each of the 500 static traces, we computed the PCC with both the corresponding interpolated trace and the corresponding desynchronized trace.

Results - After applying interpolation, the evaluation results demonstrate a significant improvement in the correlation between the traces. Fig. 6 shows a qualitative result of this evaluation procedure. The first row plots the raw desynchronized traces, the second row shows the interpolated traces at 45 MHz, and the third row plots reference static traces. It can be clearly noticed how the interpolation produces visibly similar signals to the reference static ones. Notably, the power reported in Fig. 6 refers to the voltage drop rather than the actual power consumption. Considering side-channel analysis, using of decoupled power rails is common to measure a single quantity, i.e., voltage drop, rather than both voltage and current.

Fig. 7 illustrates how the average PCC between the desynchronized traces and the static ones (green points) increases from around 0 to 0.5 after interpolation (blue points). Similarly, the standard deviation decreases from 0.2 to 0.1. These results indicate that the interpolation process successfully recovers

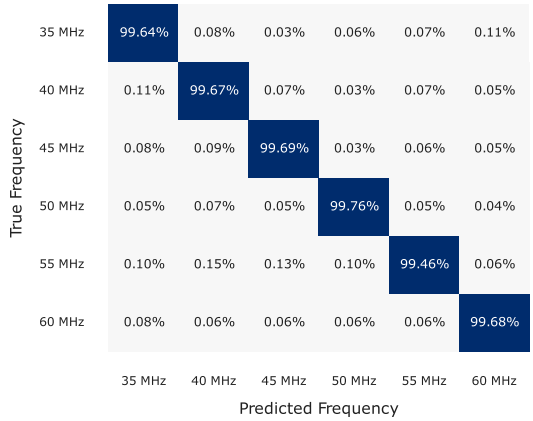


Fig. 8. Test confusion matrix of the trained CNN.

information lost due to desynchronization, making the interpolated traces more consistent and similar to the original static traces.

B. Convolutional Neural Network

This section evaluates the CNN for frequency classification illustrated in Section IV-C and details the parameters used for building the training dataset as described in Section IV-A.

Quality metrics - A Confusion Matrix (CM) describes the training evaluation results. The column indices represent the true frequencies, while the row indices represent the predicted ones. Thus, the cell CM_{ij} contains the percentage of traces whose true frequency is f_j, which the network predicted as f_i.

Configuration - The training dataset has been created starting from two 500-sample windows randomly extracted from each profiling trace in the DFS_DESYNCH database, resulting in a total of 256 000 windows. Intervals of at least 1 000 samples were considered to minimize border effects, and the center 500 samples were chosen for each window.

To avoid border effects near the frequency change, we considered intervals of at least 1 000 samples and selected the 500 samples in the center. As in standard neural networks training, we divided the collected dataset into training, validation, and test sets. Specifically, the 60% of the collected windows is used for training, while the remaining 40% is equally divided into validation and test sets.

By defining an epoch as a complete optimization step over the training set, we trained the network for a total of 8 epochs on an NVIDIA RTX™ 6000 GPU using Adam [44] to minimize the cross-entropy loss (see Equation 1). The mini-batch size was set to 128 with a learning rate of 0.001. Starting from the architecture in [36], we performed a comprehensive hyperparameter search, such as the number of layers, kernel sizes, and activation functions, using a random search strategy [45]. A separate validation set was used to evaluate the different configurations and identify the optimal combination for our specific dataset and task. The error on the validation set is evaluated after each epoch, and the final network is chosen as the one with the lowest validation error.

TABLE I
INTERSECTION OVER UNION (IOU) MEAN AND STANDARD DEVIATION COMPUTED OVER THE ENTIRE ATTACK SET OF THE DFS_DESYNCH DATABASE. TABLE REPORTS BOTH THE AVERAGED IOU VALUE AND THE IOU VALUES AGGREGATED BY FREQUENCY

	Per-Frequency IoU						Average
	35 MHz	**40 MHz**	**45 MHz**	**50 MHz**	**55 MHz**	**60 MHz**	IoU
DFT	0.9028 ± 0.0662	0.9006 ± 0.0679	0.9009 ± 0.0661	0.9234 ± 0.0532	0.8931 ± 0.0742	0.6056 ± 0.1494	0.7340 ± 0.0395
Grad-CAM	0.9944 ± 0.0008	0.9960 ± 0.0009	0.9937 ± 0.0008	0.9957 ± 0.0008	0.9958 ± 0.0009	0.9950 ± 0.0008	0.9951 ± 0.0006

Results - Fig. 8 reports the test CM at the end of the CNN training. The high concentration and diagonal dominance of the values in the confusion matrix indicate that the trained CNN effectively discriminates between different frequencies. This suggests high accuracy in the network's frequency classification task.

C. Segmentation

The core of the proposed methodology relies on the trace segmentation based on Grad-CAM (see Section IV-D). The analysis focuses on the attack set of the DFS_DESYNCH dataset.

Quality metrics - The performance of the Grad-CAM segmentation is evaluated by computing the Intersection over Union (IoU), a standard segmentation metric, between the estimated segmentations and the ones in the available ground truth. The IoU is defined by Equation 4, where P_f and GT_f are the segments corresponding to the frequency f in the predicted and ground-truth segmentation.

$$IoU_f = \frac{|P_f \cap GT_f|}{|P_f \cup GT_f|} \tag{4}$$

The numerator corresponds to the intersection between the prediction and the ground truth, while the denominator is the union. The metric measures the percentage of overlap between the segments and ranges from 0 to 1, where 1 is a perfect match between the prediction and the ground truth.

Configuration - The proposed Grad-CAM-based segmentation is compared against a segmentation method based on the analytical Discrete Fourier Transform (DFT). Both methods are evaluated by calculating the total IoU and the IoU for each frequency over the attack set of the DFS_DESYNCH dataset. The experiment considers all the 256 keys of the attack set of the database by aggregating the segmentations of the 500 traces.

Results - The proposed Grad-CAM-based segmentation perform better than the analytical DFT method across all frequencies. It also demonstrates greater stability in performance between different classes, unlike the DFT segmentation, whose accuracy varied depending on the frequency.

A significant advantage of the Grad-CAM approach is its ability to focus on the relevant frequency segments within the traces, excluding noise or irrelevant frequency components. In contrast, the DFT method is more susceptible to capturing frequencies outside the range of interest, leading to an inflated overall error rate in the IoU metric.

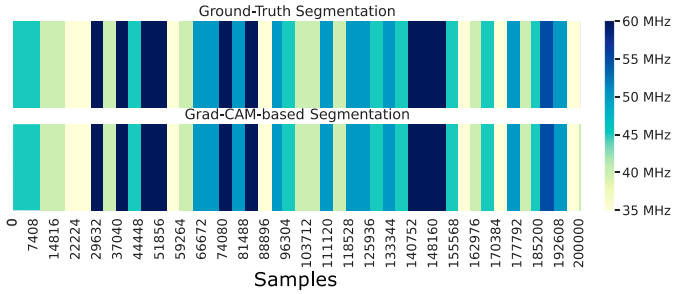


Fig. 9. Segmentation qualitative comparison of a trace from the attack set of the DFS_DESYNCH database. The first row reports the ground-truth segmentation while the second row plots the Grad-CAM-based segmentation.

TABLE II
GUESSING ENTROPY (GE) AND GUESSING DISTANCE (GD) COMPARISON OVER DFS_DESYNCH DATASET USING HAMMING WEIGHT AS LEAKAGE MODEL. ALL TEMPLATES USE PCA AS POI SELECTION METHOD

Technique	**GE**	**GD**
Raw	122.450	−0.460
DFT	109.336	−0.400
DLaTA	1.009	0.137

TABLE III
GUESSING ENTROPY (GE) AND GUESSING DISTANCE (GD) COMPARISON OVER DFS_DESYNCH AND ASCAD DATASETS USING IDENTITY AS LEAKAGE MODEL. ALL TEMPLATES USE PCA AS POI SELECTION METHOD

Technique	**DFS_DESYNCH**		**ASCAD**	
	GE	**GD**	**GE**	**GD**
Raw	130.012	−0.497	1.000	0.617
DFT	122.641	−0.481	1.000	0.617
[16]	113.895	−0.614	2.400	0.421
[27]	98.926	−0.436	1.050	0.279
DLaTA	1.000	0.255	1.000	0.617

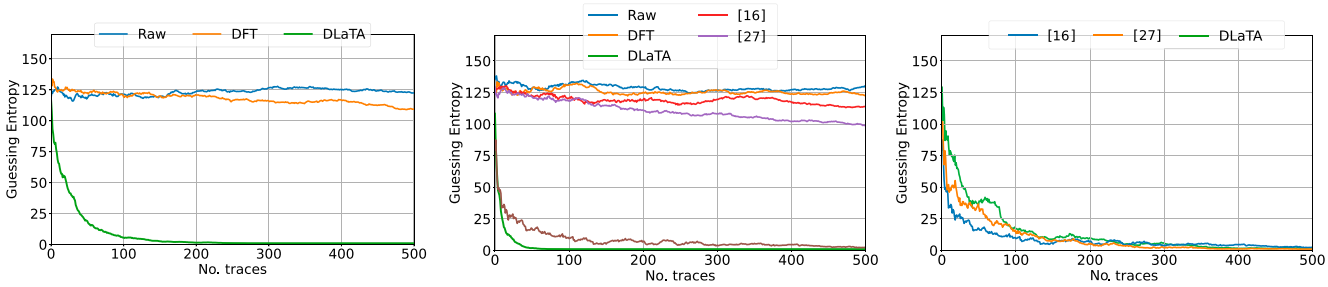
The results presented in Table I quantify this advantage. The Grad-CAM-based segmentation achieves a substantially higher average IoU of 99.40% with low variance, compared to only 73.40% for the DFT method.

A qualitative comparison between the Grad-CAM-based segmentation and the ground truth for a single trace is visualized in Fig. 9, further supporting its effectiveness for such a task.

D. Template Attack

The following presents the quality metrics used to evaluate the attack effectiveness and shows the Template Attack results obtained using the proposed DLaTA methodology. The attacked side-channel traces in the DFS_DESYNCH database represent a software AES-128 [37] encryption.

Quality metrics - Attack effectiveness is defined according to two metrics. Equation 5 defines the guessing entropy $GE(\cdot)$ as the number of key values that are more likely than the true one, averaged over all possible keys. The best possible $GE(\cdot)$



(a) Hamming Weight as leakage model over DFS_DESYNCH. (b) Identity as leakage model over DFS_DESYNCH. (c) Identity as leakage model over ASCAD.

Fig. 10. Guessing entropy results over DFS_DESYNCH and ASCAD dataset. All templates use PCA as POI selection method.

value is one, meaning the key is always precisely predicted.

$$GE(\mathcal{T}) = \mathbb{E}_{\mathcal{K}}\left[|k \in \mathcal{K} : P_T(k) \geq P_T(k^*)|\right]. \tag{5}$$

$P_T(k)$ is the probability of key k when attacking on set trace $\mathcal{T}$. Equation 6 defines the guessing distance $GD(\cdot)$ as the distance between the correct key k^* and the second guess. It is normalized by the distance between the most likely and least likely keys and averaged over all possible keys. The higher the $GD(\cdot)$ value, the more effective the attack, and negative values indicate that the correct key is not the first guess.

$$GD(\mathcal{T}) = \mathbb{E}_{\mathcal{K}}\left[\frac{P_T(k^*) - \max P_T(\mathcal{K}\backslash k^*)}{\max P_T(\mathcal{K}) - \min P_T(\mathcal{K})}\right] \tag{6}$$

Configuration - Different templates have been trained on the DFS_DESYNCH traces of Section V. The templates exploit the leakage due to the sbox substitution on the first AES round. The intermediate target value is represented by sbox(p[0] $\oplus$ k[0]), where p[0] and k[0] indicate the first byte of the plaintext and key, respectively. Those templates use 500 traces for each key value during the profiling and attack phases. PCA procedure has been applied as POI selection, keeping the five largest variance dimensions in all the performed attacks.

This work considers two leakage models: *Hamming Weight* and *Identity*. In *Hamming Weight* model, the attacker supposes the leakage is proportional to the number of active bits, as known as Hamming Weight (HW), of a chosen intermediate state, i.e., HW(sbox(p[0] $\oplus$ k[0])). In *Identity* model, the attacker assumes the leakage in the form of the real intermediate value of the cipher, i.e., sbox(p[0] $\oplus$ k[0]). Based on the leakage model, templates can be built on different classes. *Hamming Weight* has nine classes, i.e., the possible number of ones in one byte, while *Identity* has 256 classes, one for each byte value.

Three templates have been profiled for each leakage model as summarized in Tables II and III. *Raw* indicates templates built starting from raw traces, with random frequency changes during the acquisition. *DFT* shows templates built starting from a frequency segmentation based on the Discrete Fourier Transform and then scaled to 45 MHz by interpolation. Lastly, *DLaTA* illustrates templates profiled with our proposed solution, as described in Section IV, with N_{agg} set to 100. *Raw* and *DFT* templates have been repeated with different aggregation values, averaging 50, 100 or 500 consecutive samples.

Results - Tables II and III report the attack results divided by the leakage model, and it is immediately evident how the presented methodology is effective. The Guessing Entropy (GE) of raw traces, i.e., traces with dynamic frequencies, is 122.450 and 130.012 for the *Hamming Weight* and *Identity* leakage models, meaning the template has no more clue than a random guesser. A similar conclusion can be drawn for templates built on traces segmented with DFT. *Raw* and *DFT* templates have been repeated with different aggregation values, Table II reports the best results. On the other hand, traces resynchronized with the proposed DLaTA solution are clearly attackable. *Identity* leakage model (see Table III) has the best performance of all, reaching a GE value of 1.000, meaning a success rate of 100%. It is also able to properly isolate the correct key from the others with a GD of 0.255. Additionally, the *Hamming Weight* leakage model (see Table II) has a successful attack, but the slightly higher GE of 1.009 and the lower GD of 0.137 suggest how building templates on 256 classes are more effective. Fig. 10 plots the GE over the number of traces used for the attack. Guessing Entropy for raw traces shows no signs of decreasing as the number of tracks increases. Instead, the GE of the resynchronized traces reaches values close to one in less than 300 traces in both leakage models.

E. Comparison With the State of the Art

Table III also depicts a comparison between the proposed DLaTA methodology and two State-of-the-Art DL-based SCAs. All models were trained both on the DFS_DESYNCH and ASCAD [16] datasets, attacking as intermediate sbox(p[0] $\oplus$ k[0]) with 256 classes. Traces from ASCAD have been desynchronized up to 100 samples, following the random desynchronization model of the dataset. On the DFS_DESYNCH dataset, the technique in [16], which uses a CNN model, achieves a GE of 113.895 and a GD of -0.614. These results show that this technique fails to recover the secret key under high desynchronization. The second technique [27] uses a different approach that also relies on a CNN model. However, this technique also performs poorly, with a GE of 98.926 and a GD of -0.436. In contrast, our DLaTA methodology obtains a perfect Guessing Entropy of 1.000, consistently recovering the secret key. Therefore, given

the same amount of information, we can conclude that our methodology is the most effective among the three.

When we applied these methodologies to ASCAD [16], DLaTA methodology continued to outperform the others. A visual comparison of the GE over the number of traces used for the attack is reported in Fig. 10(c). The GE is close to one for all the techniques, meaning that the three methodologies are effective in recovering the secret key. However, the GD of 0.617 of DLaTA methodology is higher than the other two, showing that our methodology is more effective in isolating the correct key. Notably, ASCAD traces were collected from a computing platform operating at a single operating frequency. Therefore, DLaTA's segmentation and interpolation processes are straightforward.

VII. Conclusion

This manuscript presented a novel Deep Learning-assisted Template Attack (DLaTA) methodology for targeting desynchronized traces via DFS. The methodology consists of a pre-processing deep learning stage to resynchronize the traces to a reference clock frequency before applying a classical template attack. The pre-processing deep learning pipeline consists of three stages. The first and second stages implement a classifier and a neural network explanation tool (Grad-CAM) to identify the operating frequency associated with each sample of each desynchronized trace. Finally, the last stage implements an interpolation block to realign each trace to a reference clock frequency. The experimental results considering a software implementation of the AES cryptosystem executed on a RISC-V System-on-Chip reports a Guessing Entropy equal to 1 and a Guessing Distance greater than 0.25, thus confirming the validity of our proposal.

To promote experiment reproducibility and advance the state of the art, we have released our DFS_DESYNCH database publicly. The DFS_DESYNCH database consists of a set of real-world power traces obtained from the execution of a software version of the AES-128. The use of a random Dynamic Frequency Scaling actuator integrated into the computing platform allows to randomly desynchronize each trace.

Future works aim to extend the proposed DLaTA methodology to handle cryptosystems with Dynamic Voltage and Frequency Scaling modules and fast, fine-grained actuations over a wide range of operating points. The choice of using deep learning as a pre-processing stage followed by a Template Attack was based on the simplicity and effectiveness of Template Attacks. This two-stage methodology offers greater flexibility for the attacker, allowing them to choose the most suitable analysis technique once the side-channel measurements are resynchronized. A more powerful neural network that directly inputs a desynchronized trace and retrieves the secret key is under investigation. However, we acknowledge the need for future research to compare our method with the one in reference [7], which also combines deep learning preprocessing with Template Attack. A thorough comparison will provide valuable insights into the relative strengths and weaknesses of the two techniques.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA: Springer, 1999, pp. 388–397.

[2] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Proc. Int. Workshop Cryptogr. Hardware Embedded Syst.*, Cambridge, MA, USA: Springer, 2004, pp. 16–29.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Proc. 4th Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES)*, Redwood Shores, CA, USA, in Revised Papers, ser. Lecture Notes in Computer Science, vol. 2523, Berlin, Heidelberg: Springer, 2002, pp. 13–28.

[4] A. Heuser and M. Zohner, "Intelligent machine homicide: Breaking cryptographic devices using support vector machines," in *Proc. Int. Workshop Constructive Side-Channel Anal. Secure Des.*, 2012, pp. 249–264.

[5] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Proc. Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng.*, Hyderabad, India: Springer, 2016, pp. 3–26.

[6] A. Heuser, S. Picek, S. Guilley, and N. Mentens, "Side-channel analysis of lightweight ciphers: Does lightweight equal easy?" Cryptol. ePrint Arch., Paper 2017/261, 2017. [Online]. Available: https://eprint.iacr.org/2017/261

[7] L. Wu, G. Perin, and S. Picek, "The best of two worlds: Deep learning-assisted template attack," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2022, no. 3, pp. 413–437, 2022.

[8] G. Chiari, D. Galli, F. Lattari, M. Matteucci, and D. Zoni, "A deep-learning technique to locate cryptographic operations in side-channel traces," in *Proc. Des., Automat. Test Europe Conf. Exhib. (DATE)*, 2024, pp. 1–6.

[9] L. Goubin and J. Patarin, "Des and differential power analysis the "duplication" method," in *Proc. 1st Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES'99),* Worcester, MA, USA. Berlin, Heidelberg: Springer, 1999, pp. 158–172.

[10] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich, "Automated generation of masked hardware," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2022, no. 1, pp. 589–629, Nov. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9308

[11] S. Yang, W. Wolf, N. Vijaykrishnan, D. Serpanos, and Y. Xie, "Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach," in *Proc. Des., Automat. Test Europe*, vol. 3, 2005, pp. 64–69.

[12] J.-S. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," in *Proc. Int. Workshop Cryptogr. Hardware Embedded Syst.*, Berlin, Heidelberg: Springer, 2009, pp. 156–170.

[13] B. Hettwer, K. Das, S. Leger, S. Gehrer, and T. Güneysu, "Lightweight side-channel protection using dynamic clock randomization," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 200–207.

[14] "Advanced configuration and power interface (ACPI) specification," UEFI Forum Inc., 2022. Accessed: May 20, 2024. [Online]. Available: https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf

[15] "Raspberry Pi documentation," Raspberry Pi Foundation, 2023. Accessed: May 20, 2024. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#using-dvfs

[16] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ascad database," *J. Cryptogr. Eng.*, vol. 10, pp. 163–188, Jun. 2020.

[17] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing," in *Proc. 19th Int. Conf. Cryptogr. Hardware Embedded Syst. (CHES)*, Taipei, Taiwan. Springer, 2017, pp. 45–68.

[18] L. Batina, J. Hogenboom, and J. G. J. van Woudenberg, "Getting more from PCA: First results of using principal component analysis for extensive power analysis," in *Proc. 12th Conf. Topics Cryptol. (CT-RSA'12)*, Berlin, Heidelberg: Springer-Verlag, 2012, pp. 383–397, doi: 10.1007/978-3-642-27954-6_24.

[19] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "GRAD-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 618–626.

[20] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks vs. machine learning revisited and the curse

of dimensionality in side-channel analysis," in *Proc. 6th Int. Workshop Const. Side-Channel Anal. Sec. Des. (COSADE)*, vol. 9064. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 20–33.

[21] A. Barenghi, W. Fornaciari, G. Pelosi, and D. Zoni, "Scramble suit: A profile differentiation countermeasure to prevent template attacks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 9, pp. 1778–1791, Sep. 2020.

[22] S. Picek, A. Heuser, A. Jovic, and L. Batina, "A systematic evaluation of profiling through focused feature selection," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 27, no. 12, pp. 2802–2815, Dec. 2019.

[23] F.-X. Standaert and C. Archambeau, "Using subspace-based template attacks to compare and combine power and electromagnetic information leakages," in *Proc. 10th Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES)*, Washington, DC, USA, in ser. Lecture Notes in Computer Science, vol. 5154. Washington, D.C., USA: Springer, 2008, pp. 411–425. [Online]. Available: https://www.iacr.org/archive/ches2008/51540408/51540408.pdf

[24] B. Gierlichs, K. Lemke-Rust, and C. Paar, "Templates vs. stochastic methods," in *Proc. 8th Int. Workshop Cryptogr. Hardware Embedded Syst. (CHES),* in ser. Lecture Notes in Computer Science, vol. 4249. Berlin, Heidelberg: Springer, 2006, pp. 15–29. [Online]. Available: https://iacr.org/archive/ches2006/02/02.pdf

[25] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2019, no. 3, pp. 148–179, May 2019.

[26] M. Staib and A. Moradi, "Deep learning side-channel collision attack," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2023, no. 3, pp. 422–444, Jun. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/10969

[27] B. Hettwer, S. Gehrer, and T. Güneysu, "Deep neural network attribution methods for leakage analysis and symmetric key recovery," in *26th Int. Conf. Sel. Areas Cryptogr. (SAC)*, Waterloo, ON, Canada, in Revised Sel. Papers 26. Springer, 2020, pp. 645–666.

[28] B.-A. Dao, T.-T. Hoang, A.-T. Le, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "Correlation power analysis attack resisted cryptographic RISC-V SoC with random dynamic frequency scaling countermeasure," *IEEE Access*, vol. 9, pp. 151993–152014, 2021.

[29] F. Durvaux, M. Renauld, F.-X. Standaert, L. Van Oldeneel Tot Oldenzeel, and N. Veyrat-Charvillon, "Efficient removal of random delays from embedded software implementations using hidden Markov models," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.*, Berlin, Heidelberg: Springer, 2013, pp. 123–140.

[30] S. Nagashima, N. Homma, Y. Imai, T. Aoki, and A. Satoh, "DPA using phase-based waveform matching against random-delay countermeasure," in *Proc. IEEE Int. Symp. Circuits Syst.*, Piscataway, NJ, USA: IEEE Press, 2007, pp. 1807–1810.

[31] J. G. van Woudenberg, M. F. Witteman, and B. Bakker, "Improving differential power analysis by elastic alignment," in *Proc. Cryptogr. Track RSA Conf.*, Berlin, Heidelberg: Springer, 2011, pp. 104–119.

[32] L. Wu and S. Picek, "Remove some noise: On pre-processing of side-channel measurements with autoencoders," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2020, no.4, pp. 389–415, Aug. 2020.

[33] L. Masure et al., "Deep learning side-channel analysis on large-scale traces—A case study on a polymorphic AES," Cryptol. ePrint Arch., Paper 2020/881, 2020. [Online]. Available: https://eprint.iacr.org/2020/881

[34] J. Rijsdijk, L. Wu, G. Perin, and S. Picek, "Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2021, no. 3, pp. 677–707, Jul. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8989

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[36] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2017, pp. 1578–1585.

[37] OpenSSL, "TLS/SSL and crypto library," GitHub, 2023. Accessed: Jan. 11, 2024. [Online]. Available: https://github.com/openssl/openssl.

[38] D. Galli, A. Guarisco, W. Fornaciari, M. Matteucci, and D. Zoni, "The impact of run-time variability on side-channel attacks targeting FPGAs," 2024, *arXiv:2409.01881*.

[39] D. Galli, A. Galimberti, W. Fornaciari, and D. Zoni, "On the effectiveness of true random number generators implemented on FPGAs," in *Proc. Int. Conf. Embedded Comput. Syst.*, Samos, Greece: Springer, 2022, pp. 315–326.

[40] A. Rukhin et al., *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, vol. 22. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Washington, DC, USA, 2001.

[41] G. Scotti and D. Zoni, "A fresh view on the microarchitectural design of FPGA-based RISC CPUs in the IoT era," *J. Low Power Electron. Appl.*, vol. 9, no. 1, p. 9, 2019.

[42] D. Zoni and A. Galimberti, "Cost-effective fixed-point hardware support for RISC-V embedded systems," *J. Syst. Archit.*, vol. 126, 2022, Art. no. 102476. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122000595

[43] L. Denisov, A. Galimberti, D. Cattaneo, G. Agosta, and D. Zoni, "Design-time methodology for optimizing mixed-precision CPU architectures on FPGA," *J. Syst. Archit.*, vol. 155, 2024, Art. no. 103257. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762124001942

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations (ICLR)*, San Diego, CA, USA, Y. Bengio and Y. LeCun, Eds., 2015, *arXiv:1412.6980*.

[45] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

Davide Galli (Graduate Student Member, IEEE) received the B.Sc. degree in ingegneria informatica and the M.Sc. degree in computer science and engineering from Politecnico di Milano, in 2019 and 2022, respectively.

He is currently working toward the Ph.D. degree with Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano. Starting from his M.Sc. thesis on true randon number generators on FPGA, his research interests are mainly on hardware security and side-channel analysis.

Francesco Lattari received the B.Sc. degree in ingegneria informatica and the M.Sc. degree in computer science and engineering from Politecnico di Milano, in 2014 and 2017, respectively, and the Ph.D. degree from Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy, in 2022. Starting from his thesis on video object segmentation through deep learning approaches, his research interests are mainly on computer vision and machine learning. During the Ph.D., his research focused on the developing of machine learning approaches in the field of earth observation, by designing innovative solutions to extract information from synthetic aperture radar (SAR) satellite data.

Matteo Matteucci (Member, IEEE) received the Laurea degree in computer engineering from Politecnico di Milano, the Master of Science degree in knowledge discovery and data mining from Carnegie Mellon University, and the Ph.D. degree in computer engineering and automation from Politecnico di Milano. He is a Full Professor with the Department of Electronics Information and Bioengineering, Politecnico di Milano, Italy. His main research topics are pattern recognition, machine learning, machine perception, robotics, computer vision, and signal processing. His main research interest is in developing, evaluating, and applying, in a practical way, techniques for adaptation and learning to autonomous systems interacting with the physical world. He has co-authored more than 150 scientific international publications and he has been the principal investigator in national and international funded research projects on machine learning, autonomous robots, sensor fusion, and benchmarking of autonomous and intelligent systems.

Davide Zoni (Member, IEEE) received the Ph.D. degree. He is an Assistant Professor with Politecnico di Milano, Italy. He published more than 50 papers in journals and conference proceedings. His research interests include RTL design and verification of single- and multi-cores at the edge with emphasis on low power, hardware security, and deep learning. He filed two patents on cyber-security, and he is a Co-Founder with Blue Signals, a spin-off of the Politecnico di Milano.

Open Access funding provided by 'Politecnico di Milano' within the CRUI CARE Agreement