A Deep-Learning Technique to Locate Cryptographic Operations in Side-Channel Traces

Chiari Giuseppe
DEIB
Politecnico di Milano
Milan, Italy
giuseppe.chiari@polimi.it

Galli Davide
DEIB
Politecnico di Milano
Milan, Italy
davide.galli@polimi.it

Lattari Francesco
DEIB
Politecnico di Milano
Milan, Italy
franceso.lattari@polimi.it

Matteucci Matteo
DEIB
Politecnico di Milano
Milan, Italy
matteo.matteucci@polimi.it

Zoni Davide
DEIB
Politecnico di Milano
Milan, Italy
davide.zoni@polimi.it

***Abstract*—Side-channel attacks allow extracting secret information from the execution of cryptographic primitives by correlating the partially known computed data and the measured side-channel signal. However, to set up a successful side-channel attack, the attacker has to perform *i)* the challenging task of locating the time instant in which the target cryptographic primitive is executed inside a side-channel trace and then *ii)* the time-alignment of the measured data on that time instant. This paper presents a novel deep-learning technique to locate the time instant in which the target computed cryptographic operations are executed in the side-channel trace. In contrast to state-of-the-art solutions, the proposed methodology works even in the presence of trace deformations obtained through random delay insertion techniques. We validated our proposal through a successful attack against a variety of unprotected and protected cryptographic primitives that have been executed on an FPGA-implemented system-on-chip featuring a RISC-V CPU.**

***Index Terms*—Side-channel analysis, locating of cryptographic operations, deep-learning, computer security.**

I. Introduction

Side-channel attacks emerged as one of the most critical security threats in modern cryptography since they allow to breach into mathematically secure cryptographic algorithms by exploiting weaknesses in their physical implementation. To extract the secret key from the target implementation of the cryptographic operation (CO), side-channel attacks leverage the dependency between the data being processed and an observable environmental signal, i.e., the side-channel signal produced by the computing platform. In the last two decades, several methods, such as differential power analysis (DPA) [1], correlation power analysis (CPA) [2], template attacks (TA) [3], as well as ML-based solutions [4], [5] have been presented to maximize the effectiveness of the attack by leveraging the specific attack conditions. Notably, all the proposed techniques share two common requirements. First, they require a large number of executions of the same CO with different inputs. Second, the attacker needs to locate and align in time all the executions of the CO in the side-channel trace to feed the attack method of choice.

Considering the security assessment of a cryptographic implementation in a controlled environment, e.g., a laboratory, the attacker is granted full access to the target device. Moreover, the use of security evaluation boards, e.g., SASEBO SAKURA-II [6] and NewAE CW305 [7], makes available the so-called trigger pins that are meant to ease the alignment in time between the executions of the CO and the corresponding side-channel signals.

In contrast, the security assessment in real-world scenarios requires the attacker to perform the challenging task of locating the CO within the side-channel trace without preliminary knowledge of the location of the COs and without support of a triggering infrastructure. We note that certain real-world scenarios still allow rough alignment of the measured side-channel signal with the execution of the CO either *(i)* by leveraging specific logic events happening in the computing platform or *(ii)* by using pattern-matching techniques applied to the side-channel trace to generate so-called virtual triggers [8], [9]. Recent contributions, i.e., [10], [11], consider the identification of the COs in the side-channel trace in the presence of lightweight noise due to interrupt service routines and single-core multi-threading context switches. However, locating the COs in the side-channel trace when the computing platform implements effective randomization countermeasures, e.g., random delay [12] and dynamic frequency scaling [13], still represents a complex and open challenge.

Contributions - This work presents a deep-learning approach to locate the COs in the side-channel trace when the target device implements an effective random delay countermeasure. Our proposal presents three contributions to the state of the art:

- We present, to the best of our knowledge, the first deep-learning approach to locate the COs within the side-channel power trace in presence of a random delay countermeasure. The solution automatically locates and aligns the COs within the side-channel trace, thus allowing to mount a subsequent successful attack that is experimentally demonstrated to be impossible otherwise.
- We evaluate the proposed solution considering different cryptographic primitives and different settings for the random delay countermeasure, also performing the CPA attack to assess the quality of the achieved alignment.
- We published the tool under an open-source license, including a set of traces to allow reproducibility and further research. Tool and dataset are available at [14].

The rest of the paper is organized into four sections.

979-8-3503-4859-0/DATE24/© 2024 EDAA

Section II discusses the academic and commercial tools to locate the COs within side-channel traces. Section III presents the proposed deep-learning approach. Section IV details the experimental results. Finally, Section V presents the conclusions.

II. Related Works

Apart from the use of the so-called trigger signals made available in the security evaluation boards, e.g., SASEBO SAKURA-II [6], NewAE CW305 [7], few solutions exist to locate the COs by matching a previously computed CO template in the side-channel trace. From commercial viewpoint, Riscure icWaves [9] and NewAE ChipWhisperer Pro [8] are two FPGA-based devices offering virtual-triggering capabilities. In particular, these devices generate a trigger pulse after the real-time detection of a pattern in the monitored side-channel trace.

However, using architectural-level techniques to morph the power trace represents an easy-to-implement and effective countermeasure to deceive pattern-matching-based solutions. For example, [15] presents how time-sharing multithreading on a single-core microcontroller can hinder the attacker from correctly locating the COs, also discussing the use of interrupt service routines as a means to morph the shape of the side-channel trace. In this scenario, advanced solutions to locate the COs in the side-channel trace emerged. For instance, [10] presents a computationally efficient technique based on matched filters to locate the AES-128 cryptosystem in a power trace. The proposed technique works even in presence of interrupts that can morph the shape of the COs in the power trace. [16] proposes a waveform-matching-based triggering system that is meant to locate the COs in the side-channel trace by matching a previously computed template of the CO. [11] discusses a technique to locate the COs in a side-channel trace in the presence of interrupts and without a previously computed template of the CO.

Nevertheless, the current state-of-the-art techniques cannot locate the COs in the side-channel trace when the computing platform is protected by effective morphing countermeasures, e.g., random delay. To this end, we propose a deep-learning-based technique to locate the COs in power traces collected from a computing platform that implements the random delay countermeasure.

III. Methodology

This section overviews the developed deep-learning method for locating the COs in a side-channel trace when the computing platform implements the random delay countermeasure to effectively desynchronize the measurements. The details related to *(i)* the creation of the dataset, *(ii)* the architecture of the proposed Convolutional Neural Network (CNN) architecture, as well as *(iii)* the sliding window classifier, and *(iv)* the segmentation procedure are discussed in Section III-A, Section III-B, Section III-C, and Section III-D, respectively.

Threat model - Similarly to profiled attacks, we assume the attacker has access to an identical copy of the target device, i.e., a clone. However, we target a real-case scenario in which the attacker can only execute the applications of choice and measure the corresponding side channel without having complete and full control over the clone device. In particular, the attacker cannot activate/deactivate the desynchronization countermeasure, i.e., random delay, nor can manipulate the hardware by inserting trigger pins.

Training and inference pipelines - The proposed deep-learning methodology consists of training and inference pipelines (see Figure 1).

The goal of the *training pipeline* is to train a binary classifier that allows classifying a slice of N samples from the side-channel trace by labeling it either as *beginning of the CO* or not (see *Training Phase* in Figure 1). Notably, the methodology is meant to identify the beginning of the COs in the side-channel trace without removing the desynchronization since, once the COs are realigned, the random delay does not represent an effective countermeasure to state-of-the-art side-channel attacks [17]. We assume the attacker can use an identical copy of the target platform to create a noise trace and a set of cipher traces to feed to the training pipeline. The noise trace is a trace obtained from the execution of multiple subsequent applications different from the CO. Each cipher trace is collected during the execution of a single CO, where the attacker can choose the plaintext and the secret key. The desynchronization mechanism is active when collecting all the traces since the attacker cannot turn it off. Starting from the collected raw traces, the *Dataset Creation* block assembles a database that consists of a set of N-sample time windows extracted from the collected noise and cipher traces. For each cipher trace, the first window is labeled as *beginning of the CO*, while all the remaining windows in the trace and all the windows extracted from the noise trace are labeled as *not beginning of the CO*. Notably, the number of collected cipher traces, the length of the noise trace, and the size N of the time windows are configurable parameters. The dataset is then used to train a CNN binary classifier.

By leveraging the trained CNN, the *inference pipeline* aims at locating the COs in a new side-channel trace collected from the target device (see *Inference Phase* in Figure 1). The inference pipeline consists of three stages: *Sliding Window Classification*, *Segmentation*, and *Alignment*. The inference pipeline receives a single side-channel trace and outputs its segmentation to identify the beginning of each CO in the trace. At first, the *Slicing* block receives a side-channel trace and outputs a set of N-sample windows that are fed into the CNN classifier. The sliding amount between two consecutive and partially overlapped windows represents a configurable parameter of the proposed methodology (see s parameter in Figure 1). Starting from the classified N-sample windows, the *Segmentation* procedure outputs a vector containing the time instant locating the beginning of each CO in the side-channel trace. Finally, the *Alignment* stage cuts the initial side-channel trace according to the *Segmentation* outputs and aligns the located COs.

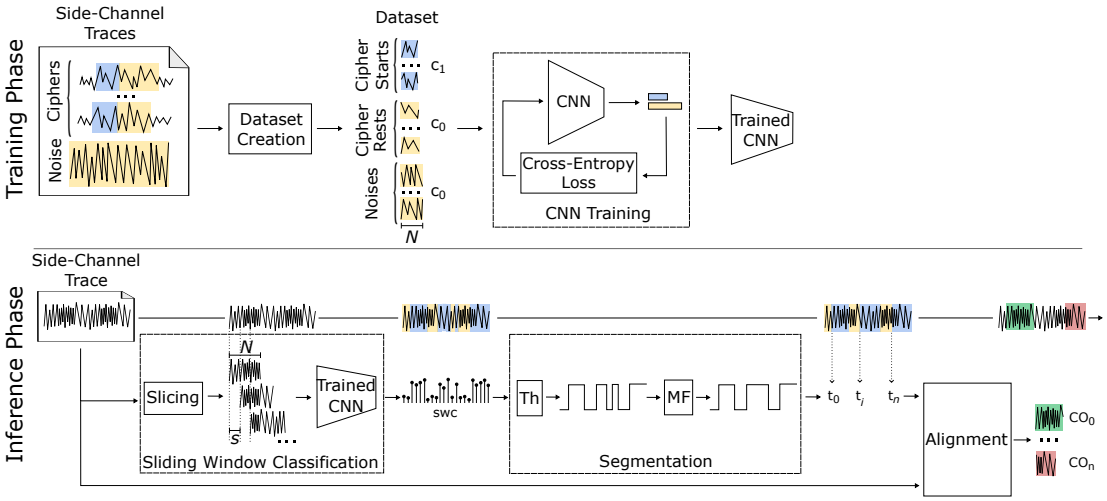


Fig. 1: Overview of the proposed pipeline for locating cryptographic operations, divided into training and inference phases. In the Segmentation block, `Th` and `MF` identify the threshold and median filter procedures.

A. Dataset Creation

Considering the training pipeline, the *Dataset Creation* block takes the side-channel traces in input and generates the dataset to train the CNN (see *Training Phase* in Figure 1). The attacker uses the identical copy of the target device to create a noise trace and a set of cipher traces. The noise trace is obtained from executing multiple applications different from the CO. Each cipher trace is collected during the execution of a single CO, where the attacker can choose the inputs. Considering the proposed threat model, the attacker can only execute a chosen application and measure the corresponding side channel on the clone device without accessing any trigger pins or deactivating the desynchronization mechanism. To build the dataset, we replace the unavailable triggering infrastructure with a set of *NOPs* instructions at the beginning of each CO. The difference in the power consumption between the execution of the *NOP* instructions and the execution of the CO allows to easily locate the beginning of the single CO in each cipher trace. Notably, the use of the NOPs in the creation of the training dataset is meant to correctly train the proposed identification infrastructure. Once trained, such infrastructure can work on the target architecture that implements the random delay without inserting any NOP instruction.

For each cipher trace i of length L_i samples, the starting N samples are labeled as *beginning of the CO* (see c_1 class in Figure 1). The remaining $L_i - N$ samples are equally split into consecutive windows of width N and labeled as *not beginning of the CO* (see c_0 class in Figure 1). Moreover, we extract a random set of N-sample windows from the noise trace and we label each of them as *not beginning of the CO* (see c_0 class in Figure 1).

B. Convolutional Neural Network

Figure 2 depicts the architecture of the proposed 1D CNN as adapted from a 2D ResNet [18]. The input to the CNN is a window ($\boldsymbol{w}$) of N samples from a side-channel trace, while its output is a classification score vector ($\boldsymbol{y}$). The CNN architecture comprises six pipelined blocks: a convolutional block, two residual blocks, a global average pooling layer, a fully connected block, and a softmax layer. Each convolutional block features a 1D convolutional layer, a batch normalization layer [19], and a ReLU activation function [20]. The residual block [18] implements two convolutional blocks enhanced with shortcut connections to sum the features element-wise (see Figure 2). Shortcut connections in convolutional blocks are used to improve the training. All the 1D convolutional layers implement a kernel with size 64, 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, while the second residual block increments the number of filters to 32.

The global average pooling layer averages the obtained features over the temporal dimension N, thus reducing the feature vector size from $N \times 32$ to 1×32. The feature vector is then fed to the two fully-connected layers with a ReLU activation function. Finally, the softmax layer outcomes a classification vector with the class scores. Notably, the structure of the global average pooling layer allows the use of different N values for the training and inference pipelines.

The error between the labels and the output of the network is computed by means of the cross-entropy loss function defined in Equation 1, where $\boldsymbol{c} \in \{0,1\}^2$ is the one-hot encoding of the class label associated to window $\boldsymbol{w}$.

$$\mathcal{L}(\boldsymbol{y},\boldsymbol{c}) = -\sum_{j=1}^{2} \boldsymbol{c}_j log(\boldsymbol{y}_j) = -\sum_{j=1}^{2} \boldsymbol{c}_j log(g_j(\boldsymbol{w},\boldsymbol{\theta})) \quad (1)$$

C. Sliding Window Classification

Considering the *inference pipeline*, the *Sliding Window Classification* block takes a new side-channel trace, slices it into N-sample windows, and uses the trained CNN to output a classification score to label each window as *beginning of the*

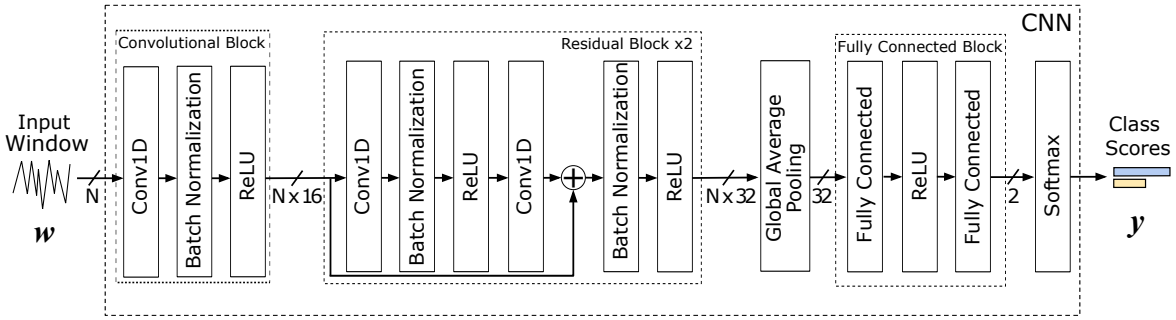


Fig. 2: Employed 1D CNN architecture. The network is an adaptation of the known ResNet [18] for 2D image classification.

CO or not (see *Sliding Window Classification* in Figure 1). The *Slicing* block implements a sliding window procedure to slice the side-channel trace in input. It takes three inputs, i.e., the side-channel trace ($trace_{inf}$), the size of the sliding window (N), and the stride (s), and outputs an ordered set of N-sample windows to feed the CNN.

The CNN outputs a new signal where each sample is a classification value. The softmax output of the CNN is a probability distribution of classes, which has been observed to hide a structured recurrent pattern that can be exploited for locating the COs. The pattern is more visible in the linear output of the fully-connected block, which makes localization easier. Thus, we consider as inference CNN output the fully-connected score outcome of class one. Depending on whether the N-sample window has been classified as the beginning of the CO or not, a higher or lower class score is assigned to the corresponding windows. Notably, the CNN can produce a noisy output that cannot be directly used to infer the precise location of the COs. To this end, the linear output of the CNN is polished by the subsequent segmentation stage in the inference pipeline (see *Segmentation* in Figure 1).

D. Segmentation

The segmentation stage takes the sliding window classification output swc and refines it to locate the beginning of the COs in the side-channel trace. It outputs a list of samples that mark the start of each CO. At first, the algorithm transforms swc into a square wave signal by comparing each sample of swc with a threshold. For each sample in swc, a corresponding output of value -1 or +1 is defined depending if the sample is below or above the threshold value, respectively (see *Th* in Figure 1). Then, a median filter (MF) is applied to further improve the accuracy of the obtained square wave (see *MF* in Figure 1). MF is fed with the square wave signal and a size k, which gives the size of the median filter window. The window slides over the fed square wave signal for which each sample is replaced with the median value of its k neighbors. At last, the algorithm returns the number of the samples identifying the rising edges, i.e., the points in the obtained square wave signal where two consecutive samples assume a value of -1 and +1, respectively. The number of the sample is multiplied by s, i.e., the stride value used during the sliding windows classification. Such points are identified as the beginning of each CO in the analyzed input trace.

TABLE I: Parameters for each pipeline stage over all the tested ciphers and dataset sizes.

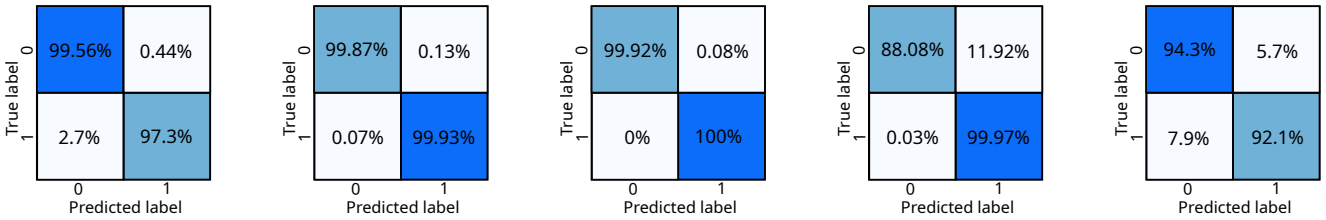
Cipher	Mean length	Pipeline Parameters			Dataset Size (N. Windows)		
		N_{train}	N_{inf}	s	Cipher Start	Cipher Rest	Noise
AES	220k	22k	20k	1k	65 536	65 536	32 768
AES mask	50k	4.8k	5k	100	131 072	65 536	65 536
Clefia	108k	6k	6k	500	65 536	32 768	32 768
Camellia	6k	1.4k	1k	100	32 768	65 536	32 768
Simon	10k	2k	2k	100	65 536	32 768	32 768

IV. Experimental Evaluation

This section discusses the experimental results of the proposed deep-learning method to locate the execution of COs in a side-channel trace while the computing platform implements the random delay as an effective trace desynchronization countermeasure. The rest of this section is organized into three parts. Section IV-A details the experimental setup. Section IV-B presents the results of the CNN training and the inference pipeline. Section IV-C discusses a complete example highlighting a successful side-channel attack, as well as the comparison with two state-of-the-art proposals.

A. Experimental Setup: Hardware and Software

As validation platform, we chose the NewAE CW305 board [7], featuring a Xilinx Artix7-100 FPGA. The power traces were collected with a Picoscope 5242d digital sampling oscilloscope (DSO), sampling at 125 Msamples/s with a resolution of 12 bits. We employed a 32-bit RISC-V System-on-Chip [21] as reference computing platform deployed on the FPGA and clocked at 50 MHz. The CPU has been modified to implement the random delay mechanism at hardware level by leveraging a true random number generator (TRNG) [22]. At run-time, the TRNG keeps generating random numbers to determine the actual number of random instructions to be inserted between each pair of consecutive program instructions. The reported results consider two random delay configurations, i.e., RD-2 and RD-4. RD-2 and RD-4 limit the maximum number of inserted random instructions between two consecutive instructions in program order to 2 and 4, respectively. As the cryptographic operation (COs) of choice,



(a) AES. (b) AES mask. (c) Camellia. (d) Clefia. (e) Simon.

Fig. 3: Test confusion matrices for the different cryptosystems affected by RD-4 random delay.

we selected the constant-time, unprotected version of four ciphers, i.e., AES-128, Camellia-128, Clefia-128, and Simon-128, from the OpenSSL software codebase [23], and a masked version of Tiny-AES-128 [24].

B. CNN Evaluation

This section focuses on the performance evaluation of the CNN. First, we detailed the dataset creation and the training metrics. Second, we reported the inference segmentation scores. A different CNN has been trained with an ad-hoc dataset for each tested cipher.

CNN training - The training of the CNN leverages an NVIDIA Titan Xp employing the PyTorch software framework. The training datasets have been collected following the procedure of Section III-A. A brief experimental campaign was carried out to find the right balance between the three window cases, i.e., *cipher start*, *cipher rest*, and *noise*. Table I reports the dataset sizes as well as the windows sizes N_{train}. Windows belonging to the ciphers, i.e., *cipher start* and *cipher rest*, are taken balanced between the key bytes. As in standard deep learning models, we divided the collected datasets into training, validation, and testing, respectively 80%, 15%, and 5% of the total.

Each network was trained for 2 epochs using Adam [25] to minimize the cross-entropy loss (see Equation 1). The mini-batches size was set to 64 with a learning rate of 0.001. The validation error was evaluated after each epoch, and the network with the lowest error was selected. As an evaluation of the goodness of the trained CNNs, their Confusion Matrices are shown in Figure 3, where for each cipher is reported the score on the RD-4 configuration. The column indices represent the true classes, while the row indices represent the predicted ones. Notably, the trained classifiers can discriminate well between the two classes, as highlighted by the high percentages on the main diagonal of each matrix.

Sliding window classification and segmentation - The pipeline parameters vary depending on the characteristics of the cipher that we want to classify, e.g., the average execution time. To this end, we experimentally determined all the required parameters. Table I shows the different values of the inference window sizes N_{inf} and the strides s, for each cipher. The use of the global average pooling layer allowed a smaller window size N_{inf} for the inference phase than the one N_{train} used for the training phase.

We measure the performance of *Sliding Window Classification* and *Segmentation* blocks by calculating the percentage of *hits*. This is the ratio of COs correctly located to the total number of true COs present in the trace. For each cipher, we tested the inference pipeline considering *i)* consecutive cipher executions and *ii)* encryptions interleaved with random applications. The execution of a sequence of consecutive COs assesses the robustness of the proposed method in locating the COs when they are executed one after the other. The execution of the COs mixed with noisy applications assesses the effectiveness of the methodology in locating the COs within a heterogeneous application scenario.

The segmentation *hits* score is 100% for every cryptographic algorithm in both scenarios, i.e., consecutive encryption and interleaved with noisy applications, always managing to find all 512 executions. The same results are achieved for both random delay configurations, i.e., RD-2 and RD-4. This demonstrates the generability of our approach, which, in addition to working with varying random delay configurations, also works on different encryption algorithms. We also show how our methodology suits protected ciphers, such as masked AES, whose side-channel traces have great variability.

C. The Complete Attack Flow

To demonstrate the effectiveness of the proposed locating method, this section presents a complete attack flow that receives an unknown side-channel trace and extracts the secret key by leveraging the Correlation Power Analysis (CPA) as the effective side-channel attack. The CPA targets the *sub-byte* intermediate. A minor aggregation over time is used to fix minor misalignments due to the rough estimation of the beginning of the COs and to mitigate the presence of random delay countermeasure.

Table II details the attack results considering different scenarios targeting the AES-128 as the CO of choice. Moreover, we compare against two state-of-the-art proposals, i.e., [26] and [11]. For each evaluated scenario, we reported the number of CO executions in the side-channel trace to achieve a rank equal to 1 in the CPA attacks for each byte of the secret key. We considered both RD-2 and RD-4 as the configuration of the random delay mechanism (see the *Random Delay Configuration* column in Table II) We considered the random interleaving of the COs within a set of noisy applications, as well as the continuous execution of the COs without any noisy

TABLE II: Segmentation and CPA results targeting AES-128. Results consider RD-2 and RD-4 settings, and the presence (or not) of noise applications interleaved with the COs.

	Random Delay Configuration	Noise Applications	Hits (%)	CPA (N. COs)
[26]	RD-2	✓	0%	✗
		✗	0%	✗
	RD-4	✓	0%	✗
		✗	0%	✗
[11]	RD-2	✓	0%	✗
		✗	0%	✗
	RD-4	✓	0%	✗
		✗	0%	✗
This Work	RD-2	✓	100%	3 695
		✗	100%	1 125
	RD-4	✓	100%	3 365
		✗	100%	1 220

application (see the *Noise Applications* column in Table II). For each analyzed scenario, our methodology can correctly identify the beginning of all the COs in the side-channel trace, leading to a successful CPA attack. In contrast, the two state-of-the-art techniques fail to locate the COs in the side-channel trace due to the random delay desynchronization, and thus, the subsequent side-channel attack is unsuccessful.

V. Conclusions

We presented a novel technique based on deep learning to identify cryptographic operations in a side-channel trace. In contrast to state-of-the-art solutions, the proposed methodology can successfully identify the COs even when the computing platform implements the random delay mechanism as the effective desynchronization countermeasure. We discussed an extensive experimental campaign to locate a series of consecutive CO executions, as well as the execution of the COs interleaved by other applications. In particular, the experimental scenarios considered several cryptographic operations and different implementations of the random delay mechanisms. The experimental results, extracted from the software execution of a different set of applications leveraging an FPGA-based RISC-V processor, confirmed the validity of the proposed methodology that allows to set up a successful CPA attack also highlighting the limitations of current state-of-the-art solutions. To facilitate reproducibility and future research, we released the tool as open-source software and provided a collection of test traces.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[2] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2523. Springer, 2002, pp. 13–28.

[4] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*. Springer, 2016, pp. 3–26.

[5] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 45–68.

[6] Inrevium Inc., "Sasebo-gii-32." [Online]. Available: https://digilent.com/reference/sasebo_g_ii_32/sasebo_g_ii_32

[7] NewAE Technology Inc., "Cw305 artix fpga target," 2018. [Online]. Available: https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA

[8] ——, "Chipwhisperer pro," https://wiki.newae.com/Tutorial_P1_Using_a_Custom_Trigger, 2020.

[9] Riscure, "icwaves," https://www.riscure.com/security-tools/inspector-hardware, 2020.

[10] A. Barenghi, G. Falcetti, and G. Pelosi, "Locating side channel leakage in time through matched filters," *Cryptography*, vol. 6, no. 2, 2022.

[11] J. Trautmann, A. Beckers, L. Wouters, S. Wildermann, I. Verbauwhede, and J. Teich, "Semi-automatic locating of cryptographic operations in side-channel traces," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, p. 345–366, Nov. 2021.

[12] F. Durvaux, M. Renauld, F.-X. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Cryptanalysis of the ches 2009/2010 random delay countermeasure," 2012, https://eprint.iacr.org/2012/038.

[13] B. Hettwer, K. Das, S. Leger, S. Gehrer, and T. Güneysu, "Lightweight side-channel protection using dynamic clock randomization," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 200–207.

[14] Hardware-Fab, "Dl-to-locate-cos-for-sca," https://github.com/hardware-fab/DL-to-locate-COs-for-SCA, 2024.

[15] I. Frieslaar and B. Irwin, "Investigating multi-thread utilization as a software defence mechanism against side channel attacks," in *Proceedings of the 8th International Conference on Signal Processing Systems*, ser. ICSPS 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 189–193.

[16] A. Beckers, J. Balasch, B. Gierlichs, I. Verbauwhede, F. Standaert, and E. Oswald, "Design and implementation of a waveform-matching based triggering system," pp. 184 – 198, 2016-01-01.

[17] 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 *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 123–140.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 06 2016, pp. 770–778.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[20] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010.

[21] G. Scotti and D. Zoni, "A fresh view on the microarchitectural design of fpga-based risc cpus in the iot era," *Journal of Low Power Electronics and Applications*, vol. 9, no. 1, 2019.

[22] D. Galli, A. Galimberti, W. Fornaciari, and D. Zoni, "On the effectiveness of true random number generators implemented on fpgas," in *International Conference on Embedded Computer Systems*. Springer, 2022, pp. 315–326.

[23] OpenSSL, "Tls/ssl and crypto library," https://github.com/openssl/openssl, 2023.

[24] MEΛITY, "Masked aes implementation," https://github.com/CENSUS/masked-aes-c, 2020.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[26] A. Barenghi, G. Falcetti, and G. Pelosi, "Locating side channel leakage in time through matched filters," *Cryptography*, vol. 6, no. 2, 2022.