2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC) | 978-1-6654-9005-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSI-SoC54400.2022.9939649

Towards Generic Power/EM Side-Channel Attacks: Memory Leakage on General-Purpose Computers

Can Aknesil
Royal Institute of Technology (KTH)
Stockholm, Sweden
aknesil@kth.se

Elena Dubrova
Royal Institute of Technology (KTH)
Stockholm, Sweden
dubrova@kth.se

***Abstract*—Today's power/EM side-channel analysis is limited by the complexity of the target hardware. We investigate the feasibility of power/EM side-channel analysis of general-purpose computers. This paper makes a step towards this goal by analyzing memory operations of Raspberry Pi 3 Model B, a widely used general-purpose IoT device that is capable of running an operating system, and shows that it is possible to extract information about the data field of memory operations from near-field EM measurements.**

***Index Terms*—Side-channel attack, electromagnetic trace, memory leakage, deep learning, Raspberry Pi.**

I. Introduction

Electronic circuits consume power and produce Electromagnetic (EM) radiation during their run-time [1]. Power/EM side-channel information can be exploited to reveal data processed at the internal states of a running system. For example, side-channel information of a mathematically secure cryptographic algorithm running on Central Processing Unit (CPU), Field-Programmable Gate Array (FPGA), or Application-Specific Integrated Circuit (ASIC) can be used to reveal the steps of the algorithm and the secret key processed by it.

Power/EM side-channel attacks performed in the past have limitations. In particular, they require the executions of the program that is targeted for analysis to be as consistent and predictable as possible through re-runs so that side-channel measurements of subsequent executions look similar, and leakage locations are the same. When enough predictability is achieved, traces do not require sophisticated post-processing, and methods, such as averaging measurements of repeated execution to increase Signal-to-Noise Ratio (SNR), are easily applicable.

This predictability can be achieved by selecting a simpler target device, like a smaller Microcontroller Unit (MCU), and having extensive control over the test environment. Faster and more complex processors introduce non-determinism to the execution behavior via, for example, dynamic branch prediction [2]. Also, it is easier to analyze programs that run directly on the processor rather than through an Operating System (OS) due to shared resources [3]. Control over the environment includes (1) the ability to restart the target device before every measurement (in [4], [5] the target FPGA is reconfigured before every measurement, in [6] the target program is wrapped by a toolchain developed for power Side-Channel Analysis (SCA)), (2) control of the physical location and orientation of the device together with stable placement of the measurement probe (measurement processes in [4], [7], [5] include opening the target chip's package, and a stable near-field probe placement to increase signal quality), and (3) ability to detect the beginning of the interesting part of the target program with good synchronization via a trigger [4], [7], [5], [3], [6]. These requirements limit the reproducibility and the generalizability of side-channel attacks.

In this paper, as opposed to previous work, we focus on SCA of individual memory operations rather than a specific cryptographic algorithm. We expect that methods that can recover information from individual memory operations would be fundamental for future SCA. These methods would enable attacks on more complex target hardware/software. Also, more specific attacks could be implemented by building on top of these methods. In other words, focusing on individual memory operations would increase the generalizability of SCA.

As a target device, a relatively complex CPU, ARM Cortex-A53, capable of running a Linux OS is selected. Memory operations of our target device take a small amount of time, requiring a higher sampling rate for capturing side-channel traces. Due to this fact, our measurements are more susceptible to the non-deterministic execution behavior of the target CPU. This susceptibility is addressed by the post-processing of the measurements.

Our contributions:

- Test Vector Leakage Assessment (TVLA) on near-field EM traces of memory operations of Raspberry Pi 3 to find leakage intensity of data read from and written to the main memory, along with the position of leakage in the traces.
- Recovery of data of memory operations using Deep Learning (DL)-based SCA.

Source code of our work is available at *https://github.com/canaknesil/rpi3-memory-leakage*.

A. Related Work

There are many examples of power/EM SCA in the literature. Some of them are, in order from recent to past: [6] performs power SCA of Advanced Encryption Standard (AES) implemented on Artix-7 FPGA; [4] performs EM SCA

978-1-6654-9005-4/22/$31.00 ©2022 IEEE

of encryption engine of Xilinx Zynq UltraScale+ FPGA; [7] performs reverse engineering of Neural Networks (NNs) through EM SCA on ATmega328P and ARM Cortex-M3 MCUs; [5] performs EM SCA of encryption engine of Xilinx 5, 6, and 7 series FPGAs.

Another work that is closely related is [3] that performs SCA on an older version of Raspberry Pi running RSA algorithm. Their work involves the identification of the steps of the RSA algorithm and the application of various analyses to traces belonging to specific parts of the algorithm in order to extract the secret key. Our work differs in that we focus on the analysis of memory operations, independent of the type of algorithm computed by the target program. This way, we expect to improve the generalization abilities of SCA.

An example of other kinds of SCA that apply to general-purpose computers is Spectre attacks [8] that exploit cache state side-channel after speculative execution. This particular attack can be mitigated via software/firmware patches. In our paper, we focus on near-field EM side-channels.

II. Background

In this section, we give a background of methods used in our experiments.

A. Pearson's Correlation Coefficient

Pearson's correlation coefficient is a normalized version of covariance. It is sensitive to the intensity of the relationship between X and Y but not to the slope.

$$r_{X,Y} = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (x_i - \overline{x})^2 \sum (y_i - \overline{y})^2}} \tag{1}$$

$r_{X,Y}$: Pearson's correlation coefficient of random variables X and Y
x_i: samples of random variable X
$\overline{x}$: mean of random variable X

The coefficient r takes a value in the interval $[-1, 1]$, 1 and -1 reflecting, respectively, maximum direct and inverse linear correlation, 0 reflecting no correlation at all. Pearson's correlation coefficient is widely used in SCA [1].

B. Test Vector Leakage Assessment

TVLA is a method used to find whether the dissimilarity of two sets of side-channel traces is statistically significant [9]. It is performed by dividing the traces into two groups where a difference is expected, and by performing t-test (see Eq. 2) across every trace point. Higher t-values indicate leakage and their positions indicate the part of the trace that is associated with the leakage.

a) Two-sample equal variance t-test: T-test is a measure of statistical significance of the difference of means of two sets.

In our TVLA, we use two-sample equal variance t-test [10]:

$$t = \frac{\overline{x} - \overline{y}}{s_p \sqrt{\frac{1}{n_X} + \frac{1}{n_Y}}}$$
$$s_p = \sqrt{\frac{(n_X - 1)s_X^2 + (n_Y - 1)s_Y^2}{n_X + n_Y - 2}} \tag{2}$$

t: t-statistic (t-value)
X, Y: sample sets
$\overline{x}$: mean of sample set X
n_X: number of samples in set X
s_p: pooled variance
s_X: variance of sample set X

C. Cross-correlation

Cross-correlation of two finite discrete signals f and g is defined by:

$$(f \star g)[m] = r_{f[n],g[n-m+N_g]},$$
$$m = 1, \ldots, N_f + N_g - 1 \tag{3}$$

r: Pearson's correlation coefficient[1]
N_f: Number of points in f.
Indices of f and g start at 1.

III. Methods

On the target device, a memory-intensive program is executed repeatedly in an infinite loop and the oscilloscope is triggered at the beginning of every iteration. EM traces are collected per execution of every loop. Traces are post-processed to extract regions in each trace that belong to memory operations. During post-processing, a trace is split into multiple traces, each belonging to one memory operation. TVLA and DL-based SCA is performed on post-processed traces to analyze the leakage and recover side-channel information.

A. Testbed

Target device: Raspberry Pi 3 Model B v1.2 [12].

- System on a Chip (SoC): Broadcom BCM2837.
- CPU: ARM Cortex-A53, 4 cores.
- Random Access Memory (RAM): 1GB LPDDR2.

Oscilloscope: Teledyne LeCroy WaveMaster 816Zi-A [13].

Near field probe: NewAE CW505 Planar H-Field Probe NAE-HPROBE-15 [14].

Amplifier: CW502 Low Noise Amplifier NPCB-LNA-02 [14].

Probe power supply: NewAE CW503 NAE-CW503-03 [14].

[1]Conventionally, dot product is used as the correlation coefficient [11]. We use Pearson's correlation coefficient instead in order to make correlation non-susceptible to amplitude scaling.

Fig. 1: Beginning of a trace before post-processing.

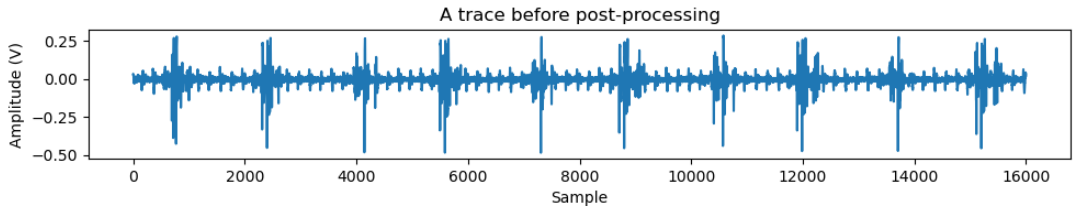
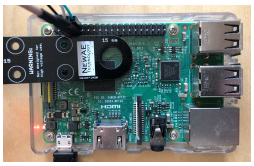


Fig. 2: Target device and EM probe.



B. Target Program

Target program is executed on bare-metal, which means directly on hardware without an intervening OS that provides a layer of abstraction [15]. Cache is disabled so that each load/store assembly instruction results in access to the main memory.

The core of the target program (the part of the program for which traces are collected) performs an array copy operation. Each store and load operation operates on 32-bit data and 32-bit address. C code and corresponding assembly code are shown in Listing 1 and Listing 2.[2]

The size of the source and the destination arrays are chosen to be 1024 ("SIZE" in the C code in Listing 1), the largest power of 2 bounded by the number of samples that could be captured by the oscilloscope. The source array is initialized with desired set of values to have control over the data field of the memory operations. The array is divided into 256 consecutive regions and all bytes of a region are initialized to the same value. The region i is initialized to the value i, for $i \in 0, 1, \ldots, 255$. The value is shifted depending on which byte of the 32-bit word is targeted during the analysis. The remaining bytes are either left as zero, or initialized at random each time the core is executed. This way all possible target byte values exist in the array and array elements with the same value are consecutive to each other.

Listing 1: array_copy in C

```
// int dst[SIZE];
// int src[SIZE];
for (int i=0; i<SIZE; i++) {
  dst[i] = src[i];
}
```

Listing 2: array_copy in assembly

```
; dst at r0
; src at r1
        mov    r2, 0x400
        cmp    r2, 0
        bxle   lr
        sub    r1, r1, 4
        sub    r0, r0, 4
        add    r2, r1, r2, lsl 2
0x8230  ldr    r3, [r1, 4]!
        str    r3, [r0, 4]!
        cmp    r1, r2
        bne    0x8230
```

[2]Edited for readability.

C. Measurement

Target program is executed in an infinite loop and a trigger is provided to the oscilloscope at the beginning of every loop, via General Purpose I/O (GPIO). EM traces are collected for each loop, each trace belonging to the entire execution of one loop.

Oscilloscope setup:

- Sampling frequency: 10 GS/s
- Bandwidth: 4 GHz

The near-field probe is placed directly on top of the CPU without opening the chip package. Target device with the probe is shown in Fig. 2. Beside the near-field probe, cables related to serial communication and oscilloscope trigger can be seen. The main memory is an external chip located at the back side of the Printed Circuit Board (PCB), approximately aligned with the CPU. We placed the probe at the front side of the chip because the front is more accessible in an attack scenario due to the fact that (1) the back side is flat and generally faces the surface the device is mounted or put on, and (2) large connectors at the front create an unused space at the center where CPU and memory chips are located.

Beginning of an example trace directly after capturing is shown in Fig. 1. Peaks belonging to first 10 memory operations can be seen.

D. Post-processing of Traces

Post-processing is performed to extract trace regions belonging to memory operations with good synchronization. Post-processing is necessary because timing of memory operations is not deterministic. Time delay between memory operations is different for each trace.

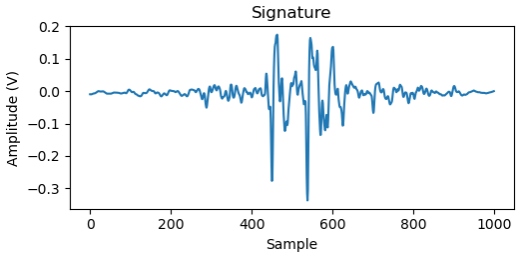
In most cases, part of the trace belonging to a memory operation has a higher amplitude and can be spotted easily as a peak (see Fig. 1). However, this is not always the case, the peak may have a smaller amplitude and/or trace may be partly noisy, making the peak difficult to distinguish.

To have a more reliable extraction, the extraction process is performed in 2 steps: (1) signature creation and (2) recognition via cross-correlation.

a) Signature creation: In this paper, the term "signature" denotes a single representative trace. A signature trace for memory operations should give an idea of what an actual trace belonging to a memory operation should look like. More technically, the correlation between a trace of a memory operation and the associated signature trace should be high.

The signature trace of memory operations is shown in Fig. 3. The signature trace is created using Algorithm 1. The signature can later be used to recognize memory operations in the traces, including the traces from which the signature was created.

Fig. 3: Signature trace for memory operations.



Algorithm 1 Create signature.

Input: One or more traces (before post-processing)
Output: The signature trace.

1) Find approximate peak locations in the input traces using Algorithm 2.
2) For each peak, extract the region of the trace having the peak.
3) Synchronize extracted regions with Algorithm 3.
4) Calculate average across the regions to create the signature.

Algorithm 2 Find peaks in a signal.

Input: A signal
Output: Location and amplitude of peaks in the signal.

1) Find local maxima above a certain threshold[3].
2) Group maxima that are closer than certain number of points and select the one with the highest amplitude in each group.

Algorithm 3 Synchronize traces.

Input: Traces
Output: Synchronized traces

1) Select one of the traces as reference.
2) For each trace except the reference trace,
   a) Calculate cross-correlation between the trace and the reference trace using Eq. 3.
   b) Shift the trace left or right according to the location of the maximum point of the correlation.

b) Recognition via Cross-correlation: Relevant regions are extracted from a non-post-processed trace using a signature with Algorithm 4. After the post-processing, one trace belongs to one memory operation.

[3]The threshold depends on amplitude of the signal and is parameterized in the source code.

Algorithm 4 Extract regions matching signature.

Input: A trace (before post-processing).
Output: Regions of the trace that correlates to the signature trace.

1) Calculate cross-correlation using Eq. 3.
2) Get peak locations in the correlation using Algorithm 2.
3) Extract regions having these peaks.

The number of memory operations that are extracted does not always match the number of expected memory operations, which can be calculated lexically from the target program. Some operations may not be extracted because of low correlation with the signature, or there may be additional memory operations because the main memory is used for other purposes such as instruction fetching. This creates the problem of memory operations belonging to the later part of the program loop having less probability of matching the extracted trace region. We match the number of extracted regions by either discarding extra regions or replicating the last region.

E. TVLA and DL-based side-channel analysis

On the target device, the main memory is 32-bit wide. TVLA and DL-based SCA are performed individually for every byte.

TVLA is performed to determine if there is side-channel leakage in traces and to find the location of leakage points.

We used DL to recover data from EM traces by categorizing traces into 256 categories representing all possible target byte values, similar to image recognition.

a) Data sets: We collected 8 trace sets. We executed 2 target programs: “array-copy” with non-targeted bytes initialized to zeros, and “array-copy” with non-targeted bytes initialized to pseudo-random values at the beginning of each program loop. We collected a different set of traces per every byte of a 32-bit word. Combinations result in 8 trace sets.

A trace set includes 2K traces before processing, 4.096M traces after processing (1024 load and 1024 store operations per non-processed trace). Each trace, after processing, includes 1K data samples. 20% of the set is left for testing. The remaining 80% is used for TVLA and DL-based SCA. During the training stage of DL models, 70% is used for training, and 30% is used for validation.

Every data set (traces and associated labels) is shuffled before splitting for test and validation.

b) DL Model and Training: We trained one Multilayer Perceptron (MLP) for every data set. We used the same architecture for every model we trained.

A model chains functions shown in Table I.

Labels are one-hot encoded.

Other hyper-parameters related to training:

- Loss function: Categorical cross-entropy
- Optimizer: NADAM [16] with learning rate 0.01
- Batch size: 64

- Training stopping condition: 20 epochs without improvement of validation accuracy.
- At the end of every epoch, the model is saved only if validation accuracy improved.

TABLE I: DL model architecture.

Function	Input size	Output size
Batch normalization	1000	1000
Dense	1000	1024
Batch normalization	1024	1024
ReLU	1024	1024
Dense	1024	512
Batch normalization	512	512
ReLU	512	512
Dense	512	256
Batch normalization	256	256
ReLU	256	256
Dense	256	256
Softmax	256	256

Python programming language [17] and TensorFlow Machine Learning (ML) library [18] are used for DL-based SCA.

IV. Results

This section summarizes the results of our experiments.

A. TVLA Results

Our results show that EM traces produced by target bytes with different Hamming Weight (HW), as well as target bytes with the same HW but different values are distinguishable.

We performed TVLA by dividing each data set in 2 ways: (1) traces with labels with HW from 0 to 3 vs. from 5 to 8, both groups having approximately 1.19M traces, and (2) trace groups with different label values with the same HW 1, in other words, a single bit in different positions, each group having approximately 12.8K traces. For the second case, we considered every pair of different single bit positions, hence $\binom{8}{2} = 28$ comparisons.

Fig. 4 and Fig. 5 show the t-test results for every 4 target bytes. The pair of target byte values 4 and 8 is chosen for plotting for the second way of division. The figures belong to two target program executions where, respectively, non-targeted bytes are initialized to zeros, and non-targeted bytes are initialized at random. The two programs executions are denoted in this section as "ntb_zero" and "ntb_random" respectively.

Table II shows the maximum point of the absolute value of the t-test vector for the comparison of HW less than vs. greater than 4, across 4 target bytes, for both program executions. Fig. 6 shows box plots of the same value, for both program executions, one box per target byte, 28 values associated to 28 comparisons in every box.

TABLE II: T-test results for $HW < 4$ vs. $HW > 4$.

	max. |t-value| (ntb_zero)	max. |t-value| (ntb_random)
byte 0	305.83	328.36
byte 1	441.85	395.22
byte 2	266.92	290.71
byte 3	239.31	245.86

B. DL-based SCA Results

An individual DL model is trained for every data set. Training and validation splits are used during training. After training, every model is tested using the associated test split.

Table III shows the testing accuracies for every model, for both target program executions

TABLE III: Empirical probability to recover a byte value from a single trace.

	byte 0	byte 1	byte 2	byte 3
ntb_zero	0.7980	0.8592	0.6486	0.8356
ntb_random	0.4922	0.7511	0.6255	0.7998

V. Discussion

The presented DL-based SCA of memory operations reveals all bytes of the operated data with accuracy from 49% to 86% with a single trace (Table III). The TVLA results in Table II and Fig. 6 show that differences in HW of labels, as well as, differences in most combinations of distinct valued labels with HW equal to 1 can be distinguished from the EM traces.

Our findings open the door to a diverse range of attacks that may be performed on Raspberry Pi 3, or similar platforms. For example, attacks that repeat the same measurement more than once and derive the final result e.g. by multiplying the resulting score vectors, or voting on the predicted labels, may be successful on our target platform.

VI. Conclusion & Future Work

In this paper, we showed that EM traces of memory operations of Raspberry Pi 3 reveal information about data read from or written to the main memory. Our TVLA shows that target values with different HW, and different target values with the same HW can be distinguished from the EM side-channel. With DL-based SCA it is possible to recover all bytes of the 32-bit data field of memory operations with accuracy ranging from 49% to 86% with a single trace.

SCA performed in this paper may be extended to incorporate enabled cache, virtual memory, the existence of an underlying OS, and other programs running in parallel on the same or other CPU cores on the same processor. Analysis of, not only data memory operations, but also instruction memory can also be interesting.

This paper is a step towards the generalization of SCA and the application of SCA to general-purpose computers. The presented results are expected to contribute to (1) better understanding of vulnerabilities of widely used general-purpose computers and related countermeasures, and (2) non-adversarial SCA of malware, e.g. ransomware.

VII. Acknowledgement

This work was supported in part by the Swedish Research Council (Grant No. 2018-0448) and VINNOVA (Grant No. 2021-02426).

Fig. 4: T-test results (non-targeted bytes set to zero).

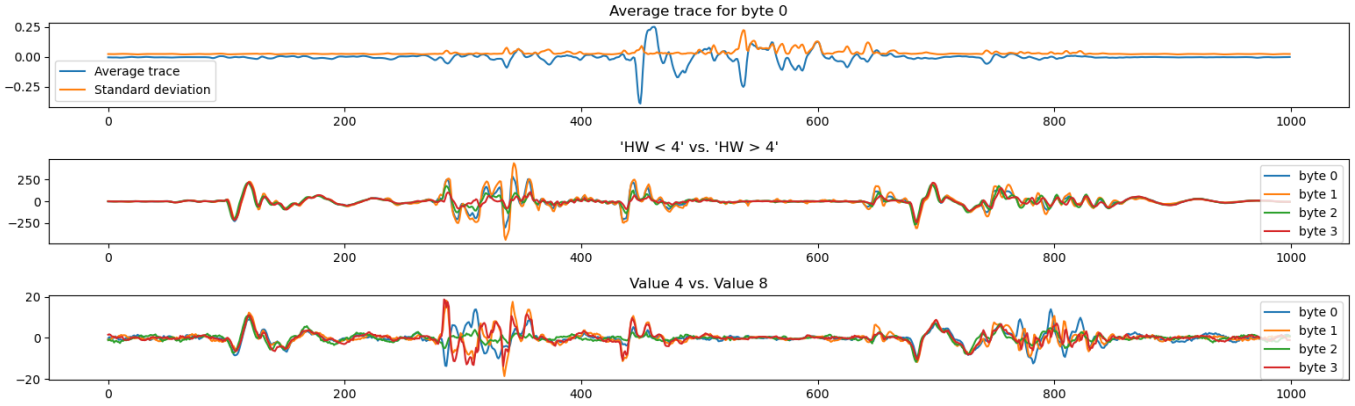


Fig. 5: T-test results (non-targeted bytes set at random).

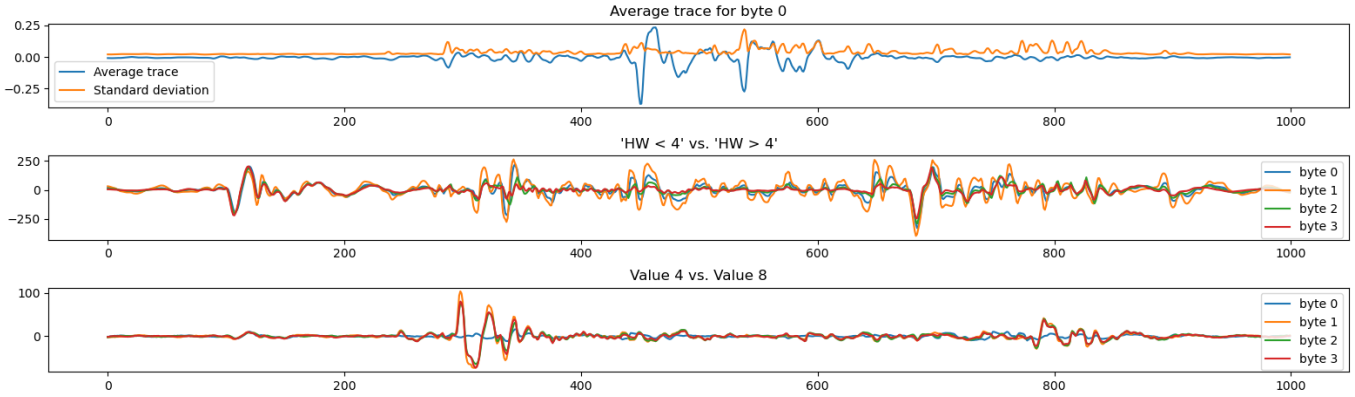
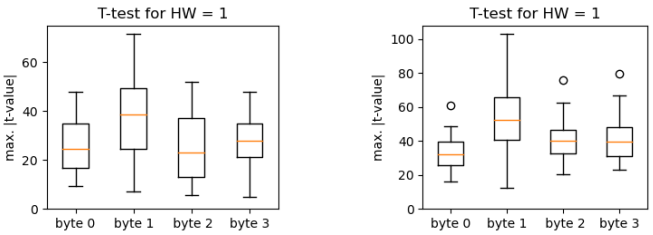


Fig. 6: T-test results for comparison of different bit positions, ntb_zero (left), ntb_random (right).



References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] C. Cheng, "The Schemes and Performances of Dynamic Branch Predictors. Berkeley Wireless Research Center," Tech. Rep., 2000.

[3] E. Hatun, G. Kaya, E. Buyukkaya, and B. O. Yalcin, "Side Channel Analysis Using EM Radiation of RSA Algorithm Implemented on Raspberry Pi," in *2019 International Symposium on Networks, Computers and Communications*, 2019, pp. 1–6.

[4] B. Hettwer, S. Leger, D. Fennes, S. Gehrer, and T. Güneysu, "Side-Channel Analysis of the Xilinx Zynq UltraScale+ Encryption Engine," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, Issue 1, pp. 279–304, 2020.

[5] A. Moradi and T. Schneider, "Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series," in *Constructive Side-Channel Analysis and Secure Design*, F.-X. Standaert and E. Oswald, Eds. Springer International Publishing, 2016, pp. 71–87.

[6] H. Wang and E. Dubrova, "Tandem Deep Learning Side-Channel Attack Against FPGA Implementation of AES," in *2020 IEEE International Symposium on Smart Electronic Systems*, 2020, pp. 147–150.

[7] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel," in *28th USENIX Security Symposium*. USENIX Association, Aug. 2019, pp. 515–532.

[8] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *40th IEEE Symposium on Security and Privacy*, 2019.

[9] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A Testing Methodology for Side-Channel Resistance Validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, pp. 115–136.

[10] "Student's t-test," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Student's_t-test

[11] "Cross-correlation," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Cross-correlation

[12] R. Ltd, "Raspberry Pi," 2022. [Online]. Available: https://www.raspberrypi.com/

[13] "Teledyne LeCroy," 2022. [Online]. Available: https://teledynelecroy.com/

[14] "NewAE," 2022. [Online]. Available: https://www.newae.com/

[15] "What is bare-metal programming? - definition from techopedia," 2022. [Online]. Available: https://www.techopedia.com/definition/3745/bare-metal-programming

[16] T. Dozat, "Incorporating Nesterov Momentum into Adam," 2016.

[17] "Python.org," 2022. [Online]. Available: https://www.python.org/

[18] "TensorFlow," 2022. [Online]. Available: https://www.tensorflow.org/