An Experimental Study on Attacking Homogeneous Averaging Processes via Side Channel Attacks

Olsan Ozbay, Yuntao Liu, *Member, IEEE*, Ankur Srivastava *Fellow, IEEE*

***Abstract*—Electromagnetic (EM) side channel attacks (SCA) have been very powerful in extracting secret information from hardware systems. Existing attacks usually extract *discrete* values from the EM side channel, such as cryptographic key bits and operation types. In this work, we develop an EM SCA to extract *continuous* values that are being used in an averaging process, a common operation used in federated learning. A convolutional neural network (CNN) framework is constructed to analyze the collected EM data. Our results show that our attack is able to distinguish the distributions of the underlying data with up to 93% accuracy, indicating that applications previously considered as secure, such as federated learning, should be protected from EM side-channel attacks in their implementation.**

I. Introduction

Side channel attacks are a powerful set of attacks that can bypass traditional security protections, and help attackers access classified information. Unlike traditional attacks, side channels do not exploit vulnerabilities in the system, but rather opt to observe the system, and learn details on how it completes certain tasks. In this paper, we will be focusing on the Electromagnetic side channel.

An Electromagnetic (EM) side channel attack (SCA) is a type of security attack that involves exploiting electromagnetic emissions from electronic systems, to gain information about their operation, or data being processed within these systems. These emissions are natural byproducts of the system functioning and adhering to user commands, thus are not directly requested by the user to be emanated.

Of the EM signals emanated from electronic systems, it is important to underline the ones belonging to computational activity, as it requires a large variety of activity in the micro-architectural level, as well as the surface level, to occur. These signals were proven significant enough to be able to determine the specific instructions that were executed in an infinite loop of code [3], which raises the question: would it be possible to know additional details of the instructions that were executed?

This paper is the summary of our attempts to extract EM side channel information of the weight of values being added in a loop. Our main contributions are:

- The setup on how it is possible to extract EM side channel data of a system running addition in a loop.
- The results of the EM side channel data collected from these systems.
- An AI model that can determine to which set of addition weights (of certain ranges) a collected EM side channel emanation sample belongs to.

Our basic premise is that information leakage occurring from EM side channels on centralized systems can be used to distinguish the magnitude of values in basic C operations, such as ADD and DIV. Thus, attackers can learn the difference of the values being inputted by observing the signal dependence of the EM side channels response to the updating code. We present a basic setup to showcase how an attacker may launch this attack on a centralized system.

Furthermore, we expand upon the limitations of the knowledge obtainable via these methods based on the results. Using these limitations in mind, we train an AI model that can automatically differentiate between different sets of homogeneous numbers that were added in the centralized system (within the limitations based off of the results).

The applicability of our measurements and the AI model is significant. Averaging of a large set of values samples from a distribution is extensively carried out in federated learning (which otherwise is touted for its security properties). In each iteration, the "players" send their local weights from running the Deep Neural Network (DNN) training on their local data to a central server which calculates the average and communicates back to the players. An EM side channel on the central server can learn the distribution of the weights being averaged in each iteration which can then be used to train the DNN by a malicious observer attacking the central server.

II. Literature Review

Over the years, side channels have proven to be a powerful class of attacks, with the EM side channel playing an important role. EM side channels can be loosely defined as where an attacker can use various tools to capture leaking electromagnetic radiation from modern computers to extract information [1] [5]. EM side channels were used to extract cryptographic keys [6], fully recover user keystrokes on wired and wireless keyboards [7], attack block ciphers [2], and much more. These potential attacks naturally sparked defenses, such as using the clock signal to mask any activity that might lead to leakage [4].

One of the more recent ways to use EM side channels to attack a system is to use the SAVAT metric: using this metric it is possible to quantify the differences in code execution, down to the level of which instruction was executed [3]. This was further expanded on, as researchers tried to quantify how much leakage certain instructions or groups of instructions would cause [8].

Since nothing more granular has been done on EM side channels, we aim to venture into this unexplored territory by trying to decipher the contents of an instruction: Trying to reverse engineer the values (floating point numbers) being used in a standard averaging process via EM side channels.

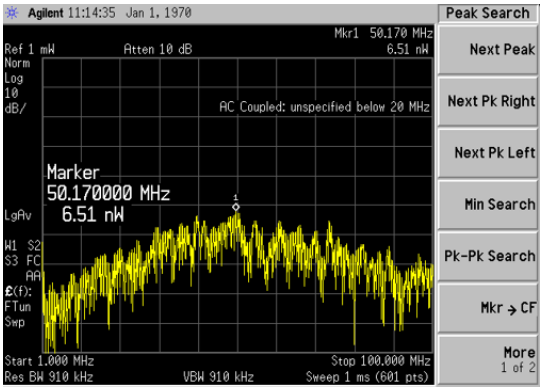

Fig. 1: Potential Spectrum Analyzer Output for One Snapshot in Time

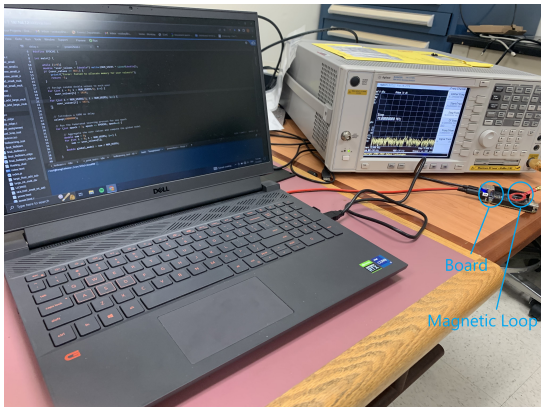

Fig. 2: Experimental Setup

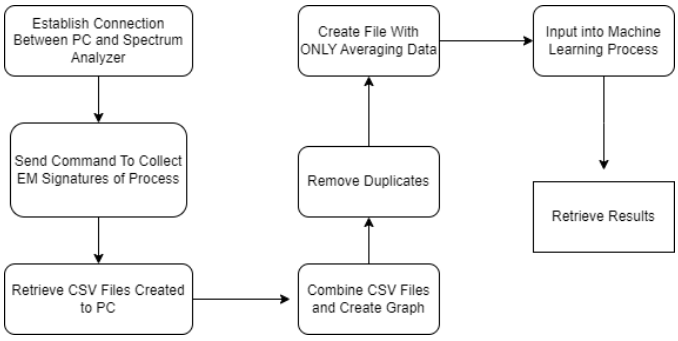

Fig. 3: Flowchart for Experiment Methodology

III. Problem Definition and Methodology

Our main goal in this research attempt was to launch an attack on a centralized computing system that was running an averaging process on a set of values. The attack would consist of collecting EM side channel data from this averaging process, and then attempting to decipher knowledge about the values corresponding to the side channel data that was collected. To create this attack, a setup involving a spectrum analyzer, magnetic loop, micro-controller single board computer (shortly referred to as "board"), and a laptop were used.

The first part of the creation of this attack involves the board, and the code for the averaging process. The sole purpose of the board was to create the victim of the attack: the board would be the centralized system that is running the averaging process which is under attack. Additionally, the code for the averaging process that was running on this board involved randomly generating 15 million numbers, averaging these numbers, and then restarting the process. A Beaglebone Black was chosen as the board for this experiment, and was not modified throughout the process.

Whilst the board ran the averaging process, the magnetic loop and spectrum analyzer were used to capture the relevant values of the EM emanations occurring. The magnetic loop was positioned within a couple centimeters of the CPU of the board, and the magnetic loop itself was attached to the spectrum analyzer. This allowed for the real time EM signals emanated from the CPU to be captured by the magnetic loop, and displayed on the spectrum analyzer's screen (which updated at a rate of 0.01 ms). The spectrum analyzer would display the values in a logarithmic decibel vs frequency graph, and an example of a potential output can be seen in Figure 1. The magnetic loop of choice was the EMRSS RF Explorer H-Loop Near Field Antenna, and the spectrum analyzer used was the Agilent E4443A.

Finally, the laptop was used to extract the EM side channel data off of the spectrum analyzer in a readable format: a CSV file. The CSV files collected are then checked for duplicates that can occur due to the spectrum analyzer lagging, and the averaging values are collected. These values are then inspected via a machine learning algorithm, which will be discussed in an upcoming section of this paper.

To summarize: The averaging process occurs on the board, which is running in an infinite loop. The attacker uses a magnetic loop which is attached to the spectrum analyzer, and positions it centimeters above of the CPU of said board to capture the EM side channel emanations. These emanated signals are then displayed on the spectrum analyzer in real time. To ensure these real time side channel values are saved, a laptop is connected to the spectrum analyzer to save the real time displayed data in CSV files. The CSV files are checked for duplicates, and the appropriate averaging values are extracted (and will be used in a machine learning process in the future). The experimental setup can be seen in Figure 2, and the flowchart for the procedure can be seen in Figure 3.

IV. Experiments & Results

A. Initial Results

The EM side channel data was collected on the averaging process of various randomized numbers. To be able to directly compare the results of the collected EM side channel, the randomized numbers in the averaging process were constrained within certain hierarchical ranges. The specific ranges for the randomized values are as follows:

- 0.0-9.9 (10^0)
- 10.0-99.9 (10^1)
- 100.0-999.9 (10^2)
- 1000.0-9999.9 (10^3)
- 10000.0-99999.9 (10^4)

TABLE I: The First 8 Rows of a Single Time Sample Output

Frequencies (MHz)	Output (log10(dB))
25.000	-68.75
25.075	-78.344
25.150	-81.466
25.225	-93.07
25.300	-84.417
25.375	-100.531
25.450	-87.018
25.525	-100.654
...	...

The constraints selected for these ranges are powers of 10, and the values themselves are floating point numbers. Once a range is selected, only random numbers from within that range can be included in the randomized number averaging process, as to preserve homogeneity, and create a clear contrast between the outputted EM side channel results. As an example: This allows for the results of the EM side channel of averaging done on values between 0.0-10.0 to be compared directly to the results of values between 10.0-99.9, to see if increasing the size of the number being averaged can directly affect the side channel output.

In the following part of this paper some key definitions are used. "**Time Sample**" or "**Sample**" for short, references a 45x1 array that is the resulting output of a single EM side channel response in time. It is 45x1 as there are 45 frequencies, and 1 time response. A "**Group**" refers to a cluster of samples, typically between 5-50 of them clustered together. Assuming 5 time samples are "grouped", the group would be an array of size 45x5: 45 frequencies, and 5 time responses.

In this experiment, over 15,000 time samples of the averaging processes were collected for each range. Once an averaging process concluded, a new one with re-randomized numbers within the set range would restart the process to ensure more samples could be collected if needed. These samples consisted of different parts of the averaging processes that were captured, as well as different averaging processes altogether (from the code restarting and new random numbers being generated). These time samples were collected into an excel sheet corresponding to their own range. An example of a singular time sample output can be seen in Table I. The rows correspond to frequencies (between 25-69 MHz), and the values within the cells are in log10(dB). Whilst this data was accurate, it was difficult to graph the values to recreate exactly what was seen on the spectrum analyzer, as the frequency sample ranges were too granular, leading to too many sudden shifts in the graph instead of the smoother, clearer images the spectrum analyzer produced. To combat this, all non-integer frequency values collected were averaged into the nearest larger integer frequency, as this was the case in the images the spectrum analyzer naturally produced. The resulting graphs were much more consistent and readable, similar to the original graphs viewed on the spectrum analyzer. An example of such an image can be seen in Figure 4.

B. Ensuring Data Quality and Grouping

After confirming the graphs were readable, and were replicating what was seen on the screen of the spectrum analyzer,

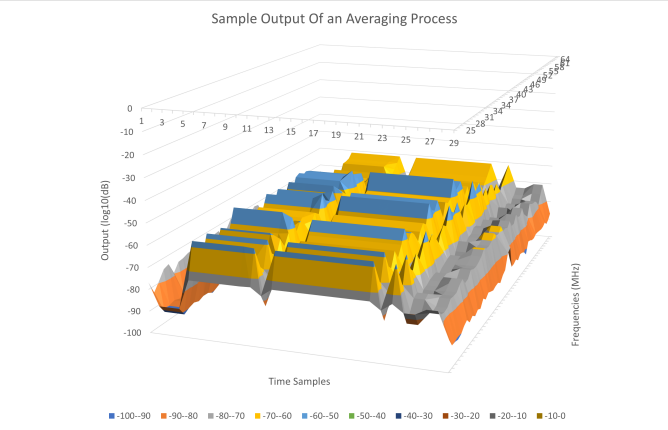

Fig. 4: Sample Output Graph of an Averaging Process

the accumulated data was put through a quality checking code. What this algorithm did was ensure that the collected time samples did not have any external noise captured that could skew the data. Common sources of noise could occur from the spectrum analyzer re-aligning, or the clock cycles changing. Additionally, if any of the components lagged, this would also duplicate the data captured across multiple time readings, so the quality checking code included a checker to omit duplicated readings.

Once the code finishes running, it creates an excel sheet with all 15,000 samples collected from the same randomized value range, where there are 45 rows for the 45 frequencies between 25 MHz and 69 MHz, and each column represents one captured data snapshot in time.

C. Differentiating the Results Manually

Whilst there was not a clear and abrupt change in the data visible in the graphs or the finalized excel sheet, there were slight changes in the maximum and minimum values obtained within the cells. It was visually apparent that as the range differences and the values within these ranges grew, the standard deviation of these numbers grew as well. To quantify this observation, the standard deviation of all the columns within each of the finalized excel sheets were taken and averaged. After averaging, an increase in the standard deviation of the values collected from most time samples were observed, but a linear increasing trend was not seen in the overall averages for each set, due to certain recorded time samples having extremely low, or extremely high standard deviations. However, when rounded to the second significant digit, the mode of the standard deviations did yield an increasing trend. This can be seen in Figure 5.

To attempt to avoid losing accuracy by rounding, and to be able to work with smaller amounts of time samples to confidently class them in a range, groups of 5-50 time samples were created. These groups would help get a grasp of the general region where the standard deviations would pile up for each range, similar to finding a direct mode. Once these groups were created, the help of machine learning was invoked.

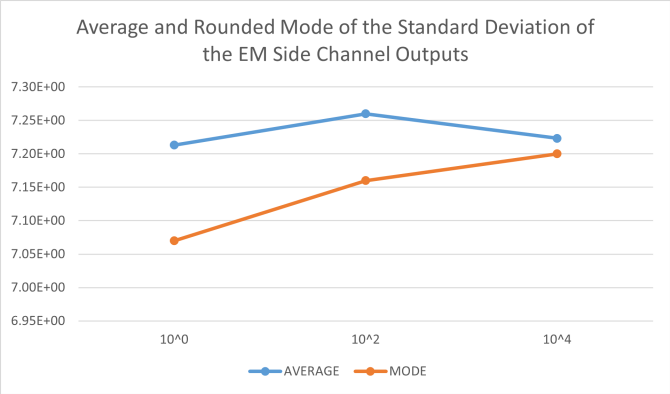

Fig. 5: Average and Rounded Mode of the Standard Deviation of the EM Side Channel Outputs

D. Differentiating the Results Using AI

The aim with using machine learning to differentiate the results is to be able to work with fewer amounts of time samples. Instead of using 15,000 samples to be able to determine which range group a sample belongs in, the AI was trained on smaller groups, from as little as 5 to as many as 50 time samples. For reference, a single averaging process takes anywhere between 14 to 20 time samples to complete. Groups larger than 50 were not tested as they forced the training sample size to be too small to produce accurate results. To further increase the chances of higher accuracy, the AI was asked to differentiate between only 2 sample ranges at a time; the inputs of sample range 10^n would be compared to the inputs of sample range 10^m, where n and m are distinct numbers between 0 and 4.

Once these sets were created, they were inputted into several different learning algorithms: Random Forest (RF), Support Vector Machine (SVM), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN). Since the largest difference in standard deviation observed in the manual inspection was between 10^0 and 10^4, these two sets were selected as the starting inputs (to be split into training and testing later on) for all of these learning algorithms.

It is also important to note that these sets were shuffled within themselves to increase homogeneity; the aim was to group the hard to distinguish samples with easy to distinguish ones to improve accuracy. Whilst this did indeed improve accuracy overall, it also increased the standard deviation achieved from the final accuracies of the machine learning process itself: The results achieved would sway by 5%+/- from their average accuracy. To combat this, all the final result of these algorithms over 10 runs were taken and averaged. This ensures the end accuracy obtained is not skewed to either end.

Whilst Random Forest and RNN failed to get more than 60% accuracy, SVM achieved on average 65% accuracy. The CNN algorithm however, surpassed all others by achieving 85% accuracy on average, with up to 93% accuracy for group sizes of 25. Thus for the testing of the other sample ranges, a group size of 25 was chosen.

Expectedly, as the difference between the range difference decreased, the accuracy also decreased. The CNN could only achieve around 60% accuracy for groups of 10^n and 10^m, where $|m-n| = 3$, and 54% for $|m-n| = 2$. For $|m-n| = 1$, it achieved even worse at 52% on average. This goes to show that while the AI is indeed useful for differentiating between larger differences in the values of the averaging process, a lot more samples are needed to achieve that same level of accuracy for smaller differences. A final table of the accuracies for the CNN algorithm can be seen in Table II. The other algorithms were omitted, as they under-performed the CNN's results in each category.

TABLE II: Average CNN Accuracies Achieved After 10 Runs (Rounded Down To the Nearest Whole Number)

—m-n— Difference	Accuracy	Algorithm
4	93%	CNN
3	60%	CNN
2	56%	CNN
1	54%	CNN

V. Applications & Future Work

Once improvements in terms of accuracy in granular cases are made, this can be a dangerous tool for attacking centralized servers working on sensitive data. An example of this would be to attack a federated learning centralized server. Centralized servers in federated learning processes are tasked with taking player models, and averaging them to create a global model. Using this improved EM side channel technique, it would be more than possible to learn about the details of the player models and the distribution of data amongst the federated learning players via the collected side channel information.

VI. Conclusion

Side channels are a powerful class of attacks that can be launched on hardware systems, and a significant amount of attacks and defenses have been proposed based on them. Recent research has shown that side channels can be assessed at the granularity of a single processor, and even individual instructions themselves [3]. This led to the idea of potentially expanding upon this granularity by being able to determine the contents within an instruction. In our case, the values and instructions used in an averaging process were chosen.

In this paper, we have managed to show that using a magnetic loop and a spectrum analyzer, we can determine the range of the values being added in homogeneous averaging processes on a micro-controller, with the only condition of the attacker being able to position the magnetic loop within centimeters of the micro-controller. We have also proven that for larger differences in the values being added, it is possible to use AI to determine which values are being added with as little as 25 time samples needed. Overall we were able to confirm which values were being averaged using the standard deviation of the values obtained from the side channel signatures.

It is important to be able to use what was found in this paper as a baseline, and improve upon the granularity and scale to launch attacks, create defenses, and further dive into the possibilities of what can be done on instruction level EM side channels.

REFERENCES

[1] D. Agrawal, B. Archambeault, et al. The EM side-channel(s). In *Proceedings of Cryptographic Hardware and Embedded Systems (CHES)*, pages 29–45, 2002.

[2] A. Bechtsoudis and N. Sklavos. Side channel attacks cryptanalysis against block ciphers based on fpga devices. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 460–461, 07 2010.

[3] R. Callan, A. Zajic, and M. Prvulovic. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 242–254, Cambridge, UK, 2014.

[4] R. Callan, A. Zajić, and M. Prvulovic. Fase: Finding amplitude-modulated side-channel emanations. *ACM SIGARCH Computer Architecture News*, 43(3S):592–603, 2015.

[5] M. G. Khun. Compromising emanations: Eavesdropping risks of computer displays. The complete unofficial TEMPEST web page, 2003. http://www.eskimo.com/~joelm/tempest.html.

[6] E. De Mulder et al. Electromagnetic analysis attack on an fpga implementation of an elliptic curve cryptosystem. In *EUROCON 2005 - The International Conference on "Computer as a Tool"*, pages 1879–1882, Belgrade, Serbia, 2005.

[7] M. Vuagnoux and S. Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *USENIX security symposium*, volume 8, pages 1–16, 2009.

[8] B.B. Yilmaz, R.L. Callan, M. Prvulovic, and A. Zajić. Capacity of the em covert/side-channel created by the execution of instructions in a processor. *IEEE Transactions on Information Forensics and Security*, 13(3):605–620, 2017.