DEEP LEARNING FOR MINIMAL-CONTEXT BLOCK TRACKING THROUGH SIDE-CHANNEL ANALYSIS

L. Jensen[⋆] *G. Brown*[⋆] *X. Wang*[⋆] *J. Harer*[⋆] *S. Chin*[⋆]

[⋆] Boston University

ABSTRACT

It is well known that electromagnetic and power side-channel attacks allow extraction of unintended information from a computer processor. However, little work has been done to quantify how small a sample is needed in order to glean meaningful information about a program's execution. This paper quantifies this minimum context by training a deep-learning model to track and classify program block types given small windows of side-channel data. We show that a window containing approximately four clock cycles suffices to predict block type with our experimental setup. This implies a high degree of information leakage through side channels, allowing for the external monitoring of embedded systems and Internet of Things devices.

***Index Terms*—** block tracking, side-channel analysis, machine learning, deep learning

1. INTRODUCTION

Computer processors consume power and emit low energy electromagnetic (EM) radiation according to the currents in their transistors and circuitry. Side-channel attacks utilize this unintended, external information to infer the current state of the processor. These attacks were first used to break cryptographic systems by analyzing the timing of processes to gain information about security keys [1]. Since then, additional studies have demonstrated that power [2], sound [3], temperature [4], and EM [5] outputs can all lead to viable side-channel attacks.

Side-channel attacks are frequently aimed at extracting private cryptographic secrets. Information from side channels can also be used in less malicious ways. As a defensive measure, analysis of side-channel signals can be used to classify machine states for malfunction detection. This is especially important on embedded systems, which are becoming more common in dedicated Internet of Things applications. Because of the specialized nature of these systems, they are frequently low in memory and processor power. As a result, traditional defensive methods such as on-device malware detection and self-monitoring are infeasible for ensuring security and maintenance. However, external malware detection and monitoring for these small systems can be achieved through side-channel analysis. Ideally, a monitoring system could detect both whether and when any deviation from a program's normal execution occurred. This requires a monitoring system to be able to track program execution.

A program's execution can be viewed as consisting of basic blocks separated by control flow, with each basic block containing a portion of assembly level instructions within which no branch or jump occurs. As a first step towards the monitoring of embedded systems, we aim to use side-channel analysis to track a program's sequence of basic blocks. We measure both the power consumption and EM radiation surrounding the target device, and then use a convolutional neural network (CNN) to segment the measured EM signals according to labelled block types at each observed time sample. In particular, we aim to show that these block types can be classified after observing side channels for only a very small number of clock cycles—showing that even these small windows of side-channel data still carry relevant leaked information.

2. RELATED WORKS

Problems in side-channel analysis include cryptanalysis, program-level classification, block-level tracking, and instruction-level tracking. Many papers have demonstrated the ability of machine learning algorithms to tackle these problems through EM and power side channels [6, 7, 8, 9, 10, 11]. Support vector machines have successfully been used for both cryptanalysis [7, 10] and program classification [6]. A handcrafted machine learning algorithm based on hidden Markov models has also been successfully applied to the sequencing of program execution at the instruction level [11].

Deep learning models have also been repeatedly demonstrated to successfully analyze the EM and power side channels. CNN models are frequently used for side-channel cryptanalysis [7, 8], and recurrent neural network (RNN) models have also been used for cryptanalysis [7]. We build upon the success of CNN and RNN models in this domain in order to implement our block-tracking model.

Deep learning models have been used less frequently for applications of side-channel analysis outside of cryptanalysis. A multilayer perceptron model was used for program classi-

Thanks to DARPA for funding this research.

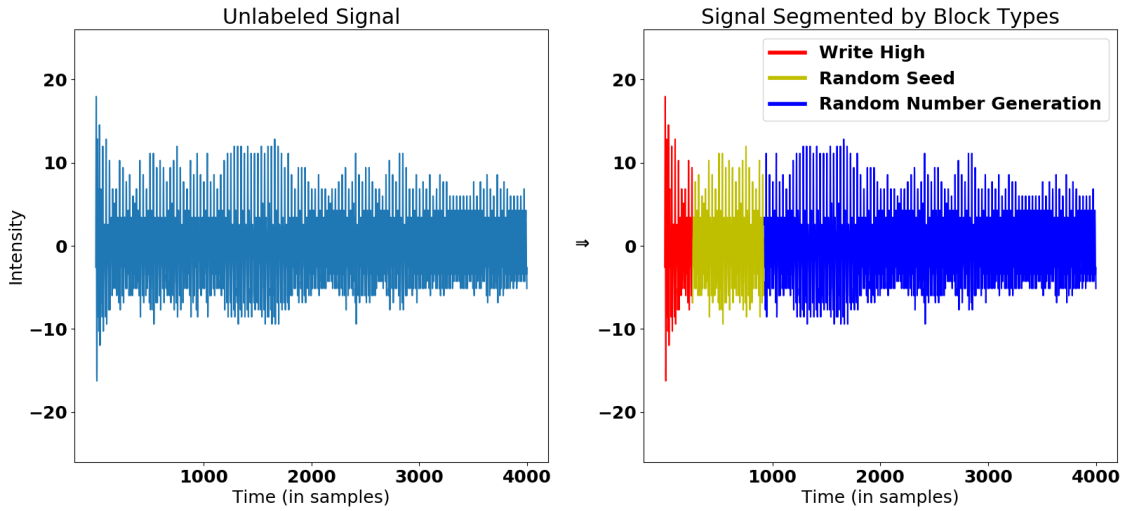

Fig. 1. Example trace and correct block type classifications for each time sample. The colors in the figure on the right represent different block classes at those time samples. Note that only a fraction of a complete program trace is shown above.

fication [9] with good results. That model analyzed the same experimental traces we analyze in this paper, but used program labels instead of the block labels we use to approach the more difficult problem of block tracking.

As side-channel analysis becomes a topic of growing interest, these applications of machine and deep learning techniques have achieved many important results in this domain. To the best of the authors' knowledge, none of these works have explored the minimum bounds of context needed for analysis. The capacity of the EM and power side channels has been studied in specific cases designed to understand the most vulnerable instructions [12, 13, 14]. Instead of focusing on specific components of the program, we test the minimum required context for block tracking across programs similar to those in real-world applications.

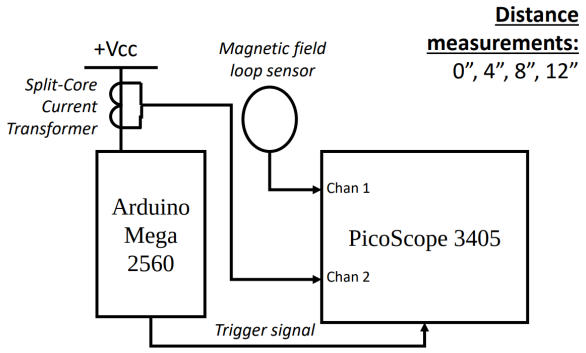

Fig. 2. The two-channel data collection setup measures power consumption and EM radiation at 500 MHz.

3. DATA COLLECTION

Our data was generated on an Arduino Mega 2560 processor, which has a clock speed of 16 MHz. The time samples were measured at a rate of 500 MHz—providing roughly 31 samples per instruction. The samples were gathered on a PicoScope 6405, with each time sample containing two simultaneous measurements: power consumption and EM radiation. We measure power consumption by measuring direct current drain from the power source, and EM radiation by measuring the magnetic field strengths. The direct current measurements were taken via an ETS-Lindgren 94430 Split-Core Current Transformer. For distance measurements, the magnetic field was measured via an Aaronia MDF 9400 broadband magnetic field tracking antenna. For close measurements, a custom version of Beehive's 100A EMC Probe was pressed to the top of the processor.

Experiments were run under four different physical setups to test signal degradation over increasing distances. The "ideal" 0" samples were gathered with a magnetic probe directly on the processor and the current transformer placed after the power regulator. The 4", 8", and 12" measurements were gathered with the magnetic antenna at those distances from the processor. For these measurements, the current transformer was at a single fixed location before the regulator on the main power connector.

Signals were measured for two different programs: math and bit-toggle. The math program repeatedly generates random numbers and was labelled with five block types: write-low, write-high, random number seed, random number generation, and loop.

The bit-toggle program repeatedly flips the register between 0x00000000 and 0xffffffff. It was labeled with three block types: write-low, write-high, and loop.

4. METHODS

4.1. Data Processing

During data collection we extracted the raw data as signal traces. Each time step of the traces was then labelled by block type using the program execution, i.e. write-low, write-high, etc. We first separated the data according to the 0", 4", 8", and 12" distances used in our data collection setups. We then subsequently separated the traces in each of these datasets into a training set of 2800 traces and a testing set of 180 traces. Each math program trace contained approximately 57,000 labeled samples and each toggle program trace contained approximately 9,000 samples. During training, we randomly sampled windows from the training traces with a bias to balance an unbalanced label distribution. For testing, we sampled without bias.

In order to find the minimal context required for block tracking, we broke the traces into windows of varying length. We tested our models with window sizes of 48, 64, 96, 128, 192, and 256 samples in length. Most windows were contained in a single block type. For the windows in transitions between block types, labels were proportional to the number of samples from each block.

4.2. Deep Learning Model Architecture

In the past several years, deep learning has achieved remarkable results in domains where data is plentiful and rich in structure [15]. These domains include signal, image, and video processing. For signal processing, one-dimensional CNNs have proven very effective for extracting important signal features. Originally applied to images, CNNs learn layers of filters that are convolved with the input, and then use the output of these repeated convolutions for classification or encoding. Another powerful deep learning model is the RNN, which operates sequentially on data while updating a hidden "memory" state. This hidden state allows the RNN to relate features across time. One successful RNN technique for handling the memory update is the gated recurrent unit (GRU) [16].

Our model architecture is shown in Figure 3. It consists of four convolution layers, a single GRU layer, and a final fully-connected layer. Inputs are processed by the convolution layers, generating basic signal features. We use convolution strides of length two instead of max-pooling in order to reduce dimension. The GRU layer then runs over the output of the convolution layers. The output of the GRU layer is then passed into the fully-connected layer which outputs a block classification. We utilized exponential linear unit activation functions [17] on each layer except for the output

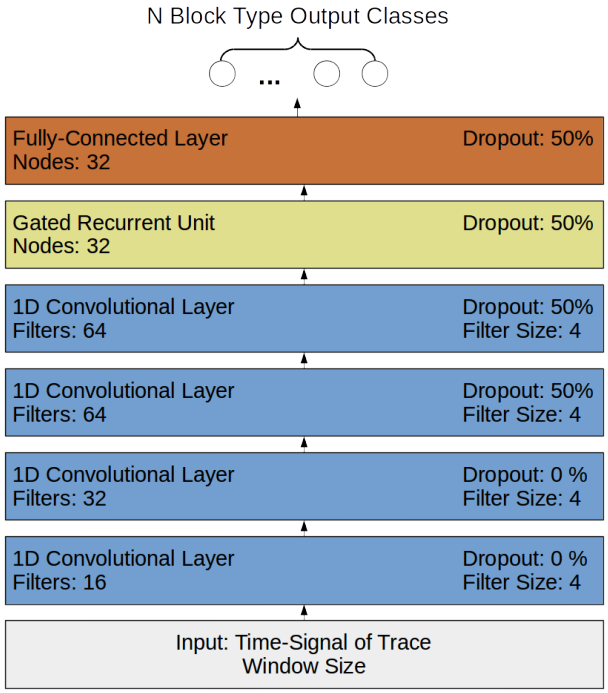

Fig. 3. Our combined CNN and GRU architecture learns basic signal features in the lower layers and combines these features across time using a gated recurrent unit. This architecture classified block types with high accuracy across many different window sizes.

layer, which uses a softmax to produce the classification. We implemented batch normalization [18] on each layer and included dropout [19] on the four last layers to combat overfitting on the training data. Models for each dataset and window size were trained from random initialization using the Keras framework [20] and optimized using the Adam optimizer [21]. Each model was trained until a pre-determined stopping epoch sufficient to ensure model convergence.

In our tests this model outperformed both models using a pure RNN applied to a spectrogram and models using fully-connected layers applied to the frequency domain. We trained an independent model for each window size and distance.

5. EXPERIMENTAL RESULTS

We trained neural networks to perform block-type classification with different window sizes for the math program. The testing results of our models trained on the math datasets are plotted in Figure 4 for all window sizes and distances. We were pleased that our model achieved high classification accuracy for multiple window sizes. However, we noted that the smallest window sizes resulted in a noticeable decrease in classification accuracy. We used this drop in accuracy to estimate a minimum window length in which the windowed data still carries sufficient information to consistently classify

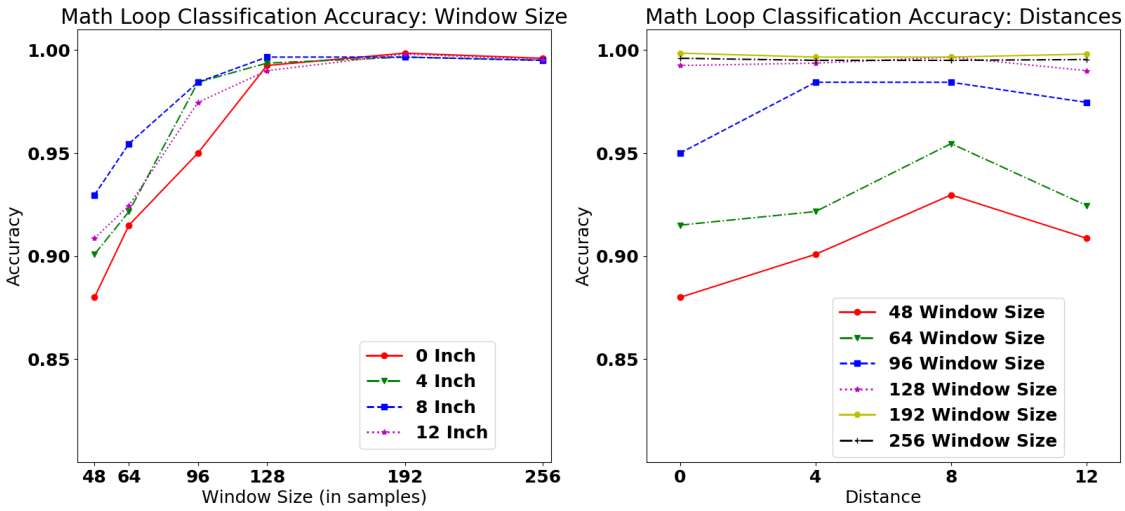

Fig. 4. Classification accuracy across various window sizes and distances. Note how accuracy plateaus above a window size of 128 samples. We estimate this window size as the minimum context required for block-type classification—equivalent to approximately four clock cycles.

block types. From our results, we estimated this minimum to be 128 sample windows, equivalent to approximately four clock cycles or 256 ns in our experiment setup.

We proceeded to verify this chosen window size using the bit-toggle program. We applied the 128 sample window size to the 0", 4", 8", and 12" bit-toggle measurements. We found that our model again achieved high accuracy on the bit-toggle program. This verifies that the 128 sample window transfers well to new program executions in the same experimental setup. Both the math and toggle program results are found in Table 1.

We did not observe any meaningful relationship between accuracy and measurement distance. This suggests that the measurement noise in this distance regime is negligible in comparison to the signal.

		Distance			
Program	Window	0"	4"	8"	12"
Math	48	87.9	90.1	93.0	90.9
	64	91.5	92.2	95.5	92.5
	96	95.0	98.4	98.4	97.5
	128	99.2	99.4	99.7	99.0
	192	99.9	99.7	99.7	99.8
	256	99.6	99.4	99.5	99.5
Toggle	128	99.7	98.3	100	99.7

Table 1. Classification accuracy for each model's performance on the distinct test sets.

6. CONCLUSION

In this paper we empirically estimate an upper bound for the minimum window context required to carry information for predicting block type. With experimental setups measuring from multiple distances, we achieve a high block-type classification testing accuracy, over 99.0%, using a window context of four clock cycles. Our results can help to estimate baseline window sizes for different experimental setups than ours (e.g. different processors, sampling rates, distances, etc.).

Many directions remain in which to extend this research. The fine-grained classification of individual blocks directly supports program-level classification. Like all supervised classification, this work is limited by the need for labelled datasets. In the future, we plan to employ unsupervised learning techniques to detect anomalies without training on explicitly-labelled datasets. We hope to recognize deviations from desired program execution such as malware or malfunction. In particular, we hope to both identify anomalies in program execution and, using the window size determined in this work, identify when in program execution these anomalies occur.

7. REFERENCES

[1] Paul Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology — CRYPTO '96*, Neal Koblitz, Ed., Berlin, Heidelberg, 1996, pp. 104–113, Springer Berlin Heidelberg.

[2] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*, Michael Wiener, Ed., Berlin, Heidelberg, 1999, pp. 388–397, Springer Berlin Heidelberg.

[3] Daniel Genkin, Adi Shamir, and Eran Tromer, "Rsa key extraction via low-bandwidth acoustic cryptanalysis," Cryptology ePrint Archive, Report 2013/857, 2013.

[4] Michael Hutter and Jrn-Marc Schmidt, "The temperature side channel and heating fault attacks," *IACR Cryptology ePrint Archive*, vol. 2014, pp. 190, 2014.

[5] Dakshi Agrawal, Bruce Archambeault, Josyula Rao, and Pankaj Rohatgi, "The em side channel(s): Attacks and assessment methodologies," 2003.

[6] Ronald Riley, James Graham, Ryan Fuller, Rusty Baldwin, and Ashwin Fisher, "Generalization of algorithm recognition in rf side channels between devices," in *Cyber Sensing 2018*. International Society for Optics and Photonics, 2018, vol. 10630, p. 106300C.

[7] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff, "Breaking cryptographic implementations using deep learning techniques," Cryptology ePrint Archive, Report 2016/921, 2016.

[8] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Canovas, "Study of deep learning techniques for side-channel analysis and introduction to ascad database," *IACR Cryptology ePrint Archive*, vol. 2018, pp. 53, 2018.

[9] Xiao Wang, Quan Zhou, Jacob Harer, Gavin Brown, Shangran Qiu, Zhi Dou, John Wang, Alan Hinton, Carlos Aguayo Gonzalez, and Peter Chin, "Deep learning-based classification and anomaly detection of side-channel signals," in *Cyber Sensing 2018*. International Society for Optics and Photonics, 2018, vol. 10630, p. 1063006.

[10] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle, "Machine learning in side-channel analysis: a first study," *J. Cryptographic Engineering*, vol. 1, no. 4, pp. 293–302, 2011.

[11] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu, "On code execution tracking via power side-channel," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2016, CCS '16, pp. 1019–1031, ACM.

[12] Robert Callan, Alenka Zajic, and Milos Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[13] Baki Berkay Yilmaz, Robert Callan, Milos Prvulovic, and Alenka Zajic, "Capacity of the em covert/side-channel created by the execution of instructions in a processor," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 605620, 2018.

[14] Alenka Zajic and Milos Prvulovic, "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 56, no. 4, pp. 885893, 2014.

[15] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[16] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. 2014, pp. 103–111, Association for Computational Linguistics.

[17] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[18] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. 2015, ICML'15, pp. 448–456, JMLR.org.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[20] François Chollet et al., "Keras," https://keras.io, 2015.

[21] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization.," *CoRR*, vol. abs/1412.6980, 2014.