Identification of Dual Processes Using Power Side-channels

Jakob Sternby, Niklas Lindskog & Håkan Englund
Ericsson Research
Lund, Sweden
e-mail: {jakob.sternby | niklas.lindskog | hakan.englund}@ericsson.com

***Abstract*—Malware is one of the main threats against electronic devices, as malicious software can damage the device, disrupt network communications and provide an entry point for additional attacks. While software-based countermeasures such as antivirus can be effective, they require presence on the device and can furthermore be disabled or fooled by advanced malware. Monitoring of physical side-channels, on the other hand, provides a non-invasive and hard-to-spoof method to detect unauthorized software being executed on a device. However, in a modern device, several processes may execute at once, making detection of alterations difficult, especially in the case where more than one process is security sensitive and should be monitored. In this paper, we present a solution for enabling granular side-channel monitoring for complex, multi-core devices. We apply new machine-learning enhanced methodology, focused on efficiently representing the measurements in latent space, to enable classification of two simultaneously executing processes. The classification training is based on labeled power side-channel traces of dual-core measurements. Our results show that it is feasible to classify two processes on separate cores having observed a single power trace obtained from a single probe.**

Keywords-Security; Side-channel Monitoring; Dual-core.

I. Introduction

Physical side-channel monitoring observes unintended information leakage from electronic devices, e.g., such as changes in the power consumption, alterations in electromagnetic fields or temperature fluctuations [1]. An important distinction, compared to classical cryptographic side-channel analysis, is that primarily data-independent architectural process leakage is observed. This approach provides a promising field for non-invasive monitoring of software processes executing on an electronic device. By analyzing these physical side-channels, an external monitor can determine which software processes are executing on a device, even without logical access to it. Figure 1 for a high level illustration of how a machine learning model is trained, and later used for inference on monitored data from a target.

However, there is still a large gap between academic literature and real world settings, preventing large scale deployments of side-channel monitors. One of the primary obstacles is the lack of research for more complex monitored environments, e.g., devices where multiple processes execute simultaneously on more than one processor or processor cores; and processor optimizations that cause non-determinism in the execution patterns. Most prior art [1][2] assumes single threaded targets. This is a reasonable assumption for low-cost embedded devices and for microcontrollers where the primary objective is to perform a very specific task. Alternatively, the assumption is made that only a single process is of interest and the rest of the processes executing should be treated as noise to filter out [3–5]. To make side-channel monitoring viable also in settings where several processes of interest execute simultaneously on different processors or processor cores it is important to overcome these hurdles. In this paper, we investigate the possibility of classifying two simultaneously executing processes, on two separate CPU cores, by measuring the power consumption of the entire device. Further, we perform the classification using a one-shot classification, i.e., the classification is done using a single power trace, without the need of repeated executions of the software. Our contribution is three-fold:

1) A multi-model machine-learning solution that improves feasibility of multiprocess side-channel monitoring.
2) An evaluation of side-channel monitoring-based classification of multiple, simultaneously executing, software processes.
3) A machine learning-based approach to classify a single side-channel measurement trace as two classes from a set of predefined processes.

The remainder of the paper is organized as follows: We describe the relevant background in Section II and discuss previous work in Section III. In Section IV, we describe the setup and properties of our solution followed by an evaluation of the effectiveness of our approach in Section V. We provide discussion of our results, as well as future work in Section VI and conclude our findings in Section VII.

II. Background

A. Side-channel emissions

Side-channel emissions refer to unintended information leaks from a physical device that occur as byproducts of its operation. These emissions can include power consumption, electromagnetic (EM) radiation, timing variations, thermal signatures, acoustic signals, and optical signals. Electronic components such as processors, memories, and data buses emit distinct side-channel data based on their current state, the instruction being executed, and the data being processed. Side-channel leakage originates from variations in power from charging and discharging transistors in hardware. The variation may both be data dependent but also depend on the set of active logic gates at the specific time. For example, power consumption can be correlated to the Hamming weight (the number of binary ‘1´s) of the current state; another common leakage mode is to consider the Hamming distance between current and previous states of the device. Historically, side-channel emissions of a device have been regarded primarily as

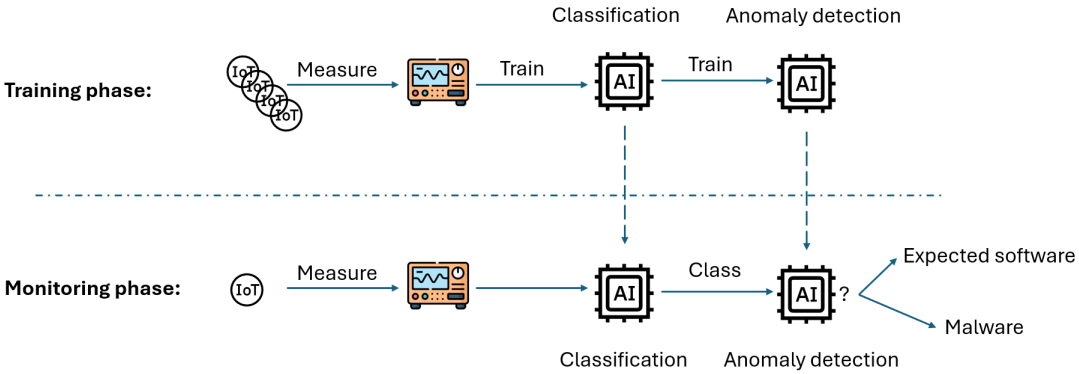

Figure 1. Overview of side-channel monitoring. Usually there is a profiling phase where e.g. a machine learning model is trained on measurements from potentially a large set of devices and varying code; and the monitoring phase where the model is used to for inference on monitored targets.

vulnerabilities that attackers could exploit to extract sensitive information [6]. An attacker can exploit the leakages to infer sensitive information in a device, e.g., to extract a cryptographic key used to encrypt data. Side-channel attacks are effective because there is a measurable correlation between the physical measurements (power consumption, EM emissions, timing, etc.) taken at various points during computation with both the data and the set of active gates of the processing device.

B. Side-channel monitoring

There is a growing interest in using side-channel monitoring as a method to detect malicious software. In this approach, a monitor records the device's side-channel emissions and determines whether its behavior aligns with predefined expected criteria. For this type of usage, called side-channel monitoring, leakage originating from data is of less relevance. Instead, the activation of components, such as different execution units and registers in a CPUs, contributes to the side-channel profile which can be used to determine the executing process. Activation patterns present in the side-channel measurements can indicate if a given process is executing on a device.

External monitoring comes with the additional advantage of not having to solely rely on information originating from processes in the device, which is a common case for system control and monitoring. A security monitoring scheme that relies on a device to itself detect deviations and non-compliances, incorporates the risk that an attacker may remain undetected, especially if he is able to mimic normality towards the control system. In this aspect, side-channel monitoring provides a compelling property of air-gapped monitoring; as the device is usually not aware if and when it is being monitored externally, stealthy malware can be detected. This is due to the fact that any unexpected process running on a device will cause an abnormal side-channel leakage.

C. Machine Learning (ML) for side-channel monitoring

Physical side-channel data, such as power consumption, can be captured as a time-series of floating point values. Depending on what is monitored, as well as the sampling rate such a time-series typically contain thousands, if not millions, of values and it would be very challenging to analyze such data without data-driven methods. In the past decade machine learning models trained on such data has surpassed prior statistically based methods when it comes to side-channel attacks [7]. Machine learning has also been an enabler for side-channel monitoring. The monitoring can include different tasks such as determining which process(es) is running as well as determining if a known process is running as expected. For determining which of a number of executing programs is running, a classification model can be trained to perform this classification based on the obtained side-channel patterns. Although a classification model produces likelihoods for their respective classes, these are not reliable to use for determining anomalies since classification models are often over-confident [8]. Training a specialized binary classifier to distinguish attacks from normal execution may be a more straightforward approach for this problem but in practice attack data is dynamic to its nature and such a model risks quickly becoming obsolete and failing to classify new attacks correctly. In anomaly or novelty detection, a model is instead trained exclusively on side-channel data from normal program executions with the aim of being able to detect anything not part of the training distribution.

Side-channel traces, or measurements, are discrete amplitude samples at regular time intervals, i.e., the data is serial. Hence, an ML-architecture with the capacity to learn correlations between different elements of sequences, such as Long-Short Term Memory (LSTM) or Recurrent Neural Networks (RNN) has been the architecture of choice for most published work on using side-channels to monitor processes [9][10]. In recent years, the Transformer architecture - a key component in the breakthrough of large language models - has shown success in a wide range of sequential ML tasks and has recently also been used with power side-channel data [11].

III. Previous Work

Side-channel monitoring using physical side-channels is well established in the literature [1]-[4][9][12]. However, to our knowledge, there is no previous attempt at dual process classification. In [5], a process is classified amid other processes executing on the device but the authors do not attempt to

classify the other processes. In [3] instruction-level disassembly is performed on a process executing on a dual core ARM Cortex-A9 CPU, however, they schedule the process on a specific core and surround the payload code with No Operation (NOP) instructions [3].

Early work on machine learning-based side-channel monitoring used a LSTM-network to classify legitimate Programmable Logic Controller (PLC) control sequences and used a threshold on the calculated softmax as an indicator of malicious code [9]. Vidal et al. use a Dynamic Time Warping with a nearest neighbor approach to align and match different power side-channel traces to classify execution blocks [2]. Classification models on power measurements have also been used to identify intrusion attacks on IoT devices. In [12], power measurements on external devices were taken every 0.2 seconds and then grouped into various feature sets. A last window of features was used to train a range of simpler models that could run on resource scarce Internet-of-Things (IoT) systems.

IV. Solution Overview

We present a solution to enable side-channel monitoring of devices with multiple processors or processors with multiple cores. We denote the monitored device as Device-under-Monitoring (DuM) for the remainder of the paper. The DuM can run several different benign processes, either distinct programs or different threads of the same program. Hence, allowing the software executing on the respective processors to be simultaneously monitored is desirable.

In the proposed solution, two processor cores execute distinct software components simultaneously on the DuM. While it would be possible to monitor the power consumption of these cores individually, it would require invasive monitoring, probing power lines to individual processors, and hardware changes. Instead, the side-channel monitor observes a physical side-channel using a single probe, e.g., monitoring the main power line of the device. One of the advantages with such side-channel monitoring is that it can be retrofitted and be entirely external to the DuM.

The monitor scans for a start trigger pattern to start measuring, e.g., a power reset indicating a boot sequence. Once the trigger is detected, the monitor collects samples until a pattern indicating an end trigger is found or a pre-defined number of samples has been collected. From the collected samples, the monitor must decide whether the measurement indicates normal behaviour or not. The solution must be able to extract this information from a set of measurements obtained during a single execution as repetition of the processes is infeasible in real-world scenarios. E.g., a boot procedure only occurs at startup and classification of processes cannot rely on obtaining measurements from multiple executions. To facilitate this, the monitor uses a machine learning-enhanced two-step approach to detect whether the processes execute as expected, as shown in Figure 1. In a first step, the monitor classifies which set of processes have been executed by the processors. In a second step, given the classified set of processes, the monitor selects a model trained for said combination and determines whether the execution was as expected.

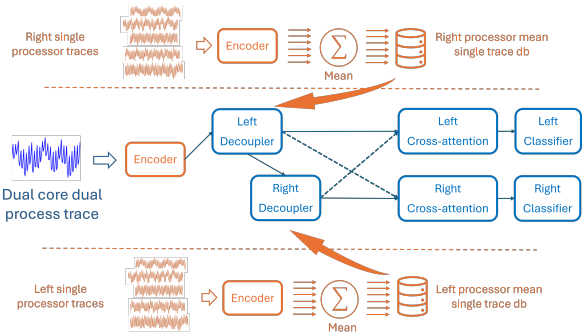

Figure 2. Overview of the ML-architecture used for dual trace classification in Section V.

In this paper, we focus on evaluating the first step, as this is a prerequisite for the second step to function. For multi-core processors which can execute several different allowed processes, effective anomaly detection cannot be performed before the monitor has determined what process it is monitoring. As it has been proven that one can monitor a known process in the presence of noise sources in a dual-core processor [3], indicating the feasibility of the second step, we aim to prove that we can identify which processes are executing. That is, the goal of our method is to use an obtained set of side-channel measurements to classify the processing class of both processors.

A. ML model architecture

The machine learning architecture used in the experiments consists of four distinct blocks: pretrained encoder, decoupler, cross-attention and classifier. A schematic overview is shown in Figure 2. Each component is based on blocks with multi-head attention, also known as transformers [13]. The purpose of the separately trained encoder is to produce a contextual representation of the side-channel traces well-suited for the classification and that this can use more and varied data without necessary labeling. The encoder is used both to convert the trace with two parallel processes running on a dual-core, as well as to convert prototype examples of a single process running on each of the cores. The encoded single process traces are used to produce static input to the decoupler modules as information of what the respective classes running on the other core could look like. Cross-attention blocks are used to mix in attention of the right and left decoupled representations to each other and finally the separated parallel classifier blocks serve to produce logits (the unnormalized probability scores) for each of the process classes of the right and left cores.

1) Pretraining a side-channel encoder with data2vec2: The encoder is trained using the data2vec2 framework [14] with a slightly modified feature encoder for side-channel measurements. Although data2vec supports multiple modalities each modality requires a specific feature encoder. As power side-channel measurements are 1D time-series similar to those of audio data, we reuse the audio encoder from data2vec with a

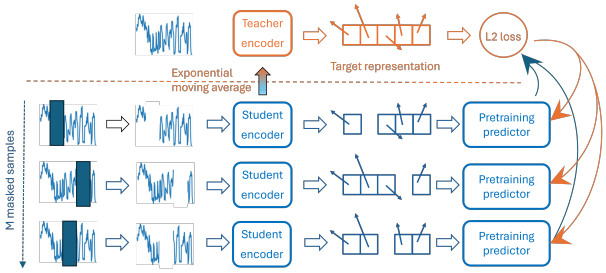

Figure 3. Overview of the pretraining process for learning side-channel latent representations.

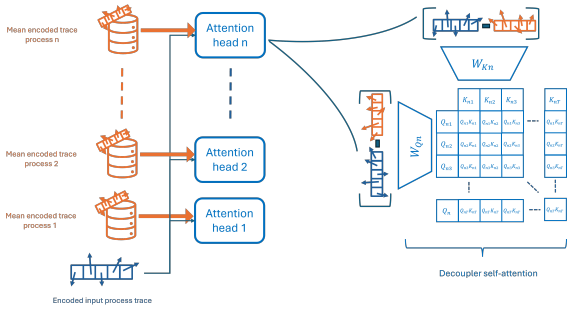

Figure 4. Overview of the self-attention mechanism in the decoupler block, incorporating the a priori knowledge of processor classes running on single cores through separate attention heads with the difference to the encoded dual trace.

stack of convolutional filters but alter the dimensions to adjust to the sampling frequency. The large share of parameters in the encoder is, however, resulting from the stack of transformer layers after these convolutional filters. The transformer layers enable the model to encode contextual information of the surrounding into each time step of the encoded sequence and the output of the encoder is commonly referred to as a contextual latent representation.

Data2vec implements a self-supervised learning paradigm where the learning task is to predict masked portions of the input sample. As depicted in Figure 3, data2vec2 employs a teacher-student configuration where the task of the student is to predict masked parts of a sample based on targets produced by the teacher. The teacher on the other hand is an exponential moving average of the student encoder. In order to enable prediction of the masked parts, the student network has an additional prediction network component which in this implementation is a stack of 1D convolutional layers. Each target representation is reused for multiple maskings of one sample for efficiency as shown in the figure. Note that the pretraining prediction network is only used during the pretraining of the encoder and is discarded afterwards.

2) Training a dual-trace classifier with single process representations: The neural network architecture presented in this paper has two major differences from a conventional transformer-based sequence classifier. Firstly, the dual-core classifier presented here has two output blocks - one for each core - to enable classifying two processes simultaneously, as seen in Figure 2. Secondly, with the aim of incorporating knowledge of the side-channel measurements of single core processes, it contains novel decoupler blocks that mix in knowledge of these encodings into separate attention heads.

The idea is that a neural network could learn to separate two processes in a combined dual-core trace if given the difference of each process in the encoded representation. To accomplish this, the decoupler block has been designed as a multi-head attention block where each attention head takes class-specific input, as seen in Figure 4. The class-specific input of a processor are the mean encoded representations of measurements for each process running on a single core. Each attention head then performs normal self-attention on the difference between the class-specific input of that head and its encoded input. By placing two decouplers, first one left with class-specific inputs from single processes running on the left processor followed by one on the right taking the output of the first decoupler and class-specific input from single processes running on the right processor, the model could learn to successively separate the processes present in the dual-core measurements.

The final output from all attention heads in one decoupler is concatenated and projected back to the input dimension. After the decouplers, the respective left and right outputs are separated and continue with cross-attention blocks which add another possibility to adjust the right and left outputs with respect to each other. The cross-attention blocks are identical to the cross-attention blocks in a typical decoder of a encoder-decoder network [13], where the key and query inputs are the separated left and right outputs from the decouplers, respectively. The two parallel classification blocks consist of multi-headed attention followed by a projection from the concatenation of the sequence to the number of classes and ending with a softmax outputting the class probabilities. All of the components after the encoder contain two identical serial blocks and the cross-attention and classification blocks have four attention-heads each in the multi-headed attention.

V. Evaluation

Our experimental setup comprised a ChipWhisperer Husky measuring power consumption on targets implemented on a DuM embodied by a NAE-CW305-04-7A100 FPGA. The FPGA was configured with two soft-core realizations of Cortex-M3 cores [15] with 32 kB instruction memory and 32 kB data memory, both implemented in block RAM. The respective cores have a three-stage instruction pipeline, which has branch prediction, and do not have a cache. Moreover, the first Cortex-M3 (denoted $\mathcal{C}_\mathcal{A}$) has a trigger signal for informing the Husky of process start. The first and second Cortex-M3 (denoted $\mathcal{C}_\mathcal{B}$) share a reset signal, and therefore start to execute their respective software programs simultaneously. The cores are executing at a clock speed of 20 MHz and the monitor samples at a rate of 80 MS/s. 4x oversampling was selected as it is sufficient to identify the patterns of the programs executed on the respective CPU core, but still low enough to generate reasonable amounts of data. The evaluation utilized programs from the BEEBS [16] suite and we selected 10

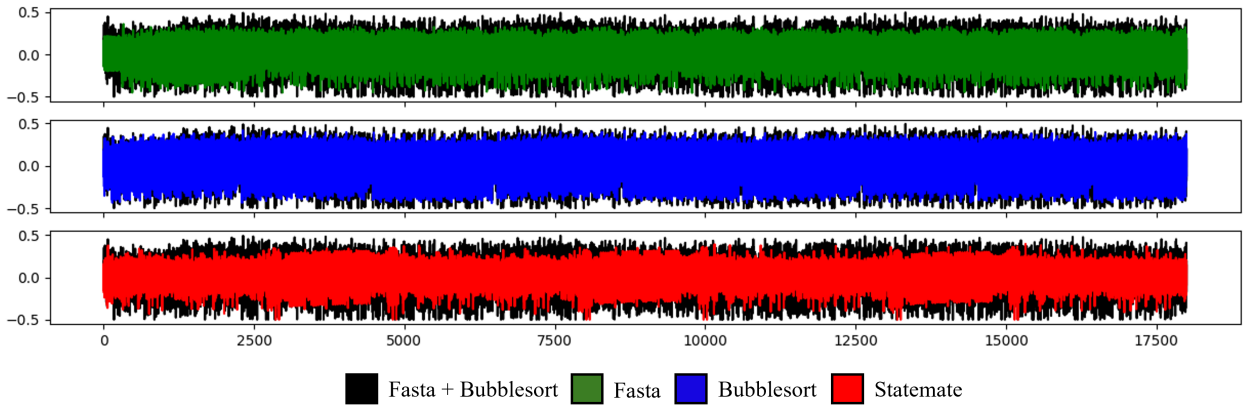

Figure 5. An example of the contribution of each process to a joint side-channel trace. Each measurement trace indicates the relative power consumption of the device during 4500 clock cycles. The black trace is a measurement of the FASTA program executing on $\mathcal{C}_\mathcal{A}$ and the BUBBLESORT program on $\mathcal{C}_\mathcal{B}$. In the green trace, the FASTA program is executing by itself on $\mathcal{C}_\mathcal{A}$, i.e., $\mathcal{C}_\mathcal{B}$ is idle. In the blue example, the BUBBLESORT program is executing by itself on $\mathcal{C}_\mathcal{B}$. In the red example, a program not present in the black trace, STATEMATE, is executing on $\mathcal{C}_\mathcal{B}$.

programs $P = \{P_i \mid 0 \leq i \leq 9\}$ of suitable execution length. The selected programs have deterministic execution e.g. without non-deterministic branch conditions. We measured each program for 18000 samples, i.e., during 4500 clock cycles. For programs shorter than 4500 clock cycles, the program was restarted. An example of the collected samples can be seen in Figure 5.

Let P^A ($P^\mathcal{A} \in P$) denote a program running on $\mathcal{C}_\mathcal{A}$, and $P^\mathcal{B}$ a program on $\mathcal{C}_\mathcal{B}$ respectively.

During the measurement collection phase, we collected 1000 measurements from each combination of programs ($\{[P^\mathcal{A}, P^\mathcal{B}] \mid \forall P^\mathcal{A}, P^\mathcal{B} \in P\}$), as well as each program running alone on respective processor. 70% of these were fed to the evaluation model during training 10% was used as a validation set and the rest as a held-out independent test set.

A. Training the ML model

The pretraining of the encoder was conducted with the data2vec framework [14] in an unsupervised manner with side-channel measurements from 10 different programs on a single-core, as well as some measurements from two processes running on a dual-core processors. For the first convolution layers of the encoder we used four layers of dimensions `(256, 256, 128, 128)`. Corresponding kernel sizes and strides of the layers were: `(32, 12, 4, 4)` and `(12, 8, 3, 2)`. The encoder dimension was 48 and in total the encoder produced 29 time steps for each trace-segment of length 18,000. The disposable 1D convolutional decoder only used in the pretraining had 4 layers with 64 dimensions, 8 groups and kernel size 7. In total, the pretraining with a total of 120120 traces ran for 120000 steps using the Adam optimizer with a learning rate of 0.0001 and weight decay of 0.01.

The dual classification model is trained without changing the pretrained model. It starts by calculating the mean of the encoder output of the single process training traces for each right and left class, these are then added as static elements in the right and left decoupler blocks seen in Figure 2. The training then proceeds with two parallel cross-entropy loss calculations of the corresponding labels for left and right processor, each contributing to updating the specific and common weights of the network. The model was trained for 140 epochs with the Adam optimizer with a starting learning rate of 0.0002 and a weight decay of 0.00003. The best model as evaluated by the loss on the validation set was chosen as the final model of each run. The seed was changed for each of 5 runs included in Table I to get a different split of the collected data into training / validation / test.

B. Classification results

The classification model is evaluated with two accuracy metrics as seen in Table I. The accuracy listed is the percentage of test traces where the complete dual label is correct whereas the *single acc.* is calculated as the percentage of single labels being correct. *Single acc.* is consequently at least as high as accuracy but usually higher since accuracy will classify an output as incorrect even when one of the two labels is correct. The results show the mean and the standard deviation of five runs with different seeds causing differing splits into training / validation and test sets. To compress the table only the aggregation of results for all processes running on the $\mathcal{C}_\mathcal{A}$ is shown $\mathcal{C}_\mathcal{B}$ in Table I.

VI. Discussion

In this paper, we have allowed the simplification of having synchronous executions by using a single reset signal to both processors. A natural next step would be to determine how well the presented solution works for programs which are slightly displaced in time, as the assumption of deterministic process start is only viable in very specific situations.

The Cortex-M3 processors does not have any cache and thus no shared state. The co-varience of processes with shared caches should be studied further.

Furthermore, we have only considered simple processes with deterministic execution, without branching. Combining

TABLE I. DUAL-CORE CLASSIFICATION ACCURACY LISTED BY THE CLASS OF THE PROCESS RUNNING ON $\mathcal{C}_\mathcal{A}$.

$\mathcal{C}_\mathcal{A}$ class	Accuracy	Single acc.	# Test	# Valid	# Train
cnt	$96.1\% \pm 0.9$	$98.0\% \pm 0.5$	2200	1100	7700
fasta	$91.1\% \pm 1.1$	$95.6\% \pm 0.5$	2000	1000	7000
prime	$97.5\% \pm 0.4$	$98.8\% \pm 0.2$	2200	1100	7700
ahacompress	$95.8\% \pm 4.8$	$97.9\% \pm 2.4$	2200	1100	7700
bubblesort	$97.3\% \pm 4.2$	$98.6\% \pm 2.1$	2200	1100	7700
cover	$98.2\% \pm 4.1$	$99.1\% \pm 2.0$	2200	1100	7700
tarai	$91.9\% \pm 2.0$	$95.9\% \pm 1.0$	2200	1100	7700
lcdnum	$96.3\% \pm 3.6$	$98.2\% \pm 1.8$	2200	1100	7700
crc32	$85.4\% \pm 3.6$	$92.7\% \pm 1.8$	2200	1100	7700
statemate	$96.4\% \pm 5.0$	$98.2\% \pm 2.5$	2200	1100	7700
idle[a]	$88.5\% \pm 5.0$	$94.2\% \pm 2.5$	2200	1100	7700
Total	$94.1\% \pm 2.2$	$97.0\% \pm 1.1$	24000	12000	84000

[a]No process currently executing on $\mathcal{C}_\mathcal{A}$.

our multi-process work with control flow graphs has potential to enable monitoring of the current internal state of multiple processes simultaneously.

In this paper, we have performed our tests on an FPGA implementation of the two CPUs. This has enabled us to make simplifications in order to fine-tune the methodology. How additional complexity impacts the results, in the form of executing the software on hard processors, with or without out-of-order execution and processor optimizations, should be researched further.

VII. CONCLUSION AND FUTURE WORK

We showed that two distinct processes execution on two separate ARM Cortex M3 processors can be correctly classified with an average accuracy of 94.1%. This indicates that the side-channel monitoring can adapt to more complex devices and that monitoring can be viable also for use cases going beyond to single-process CPUs and FPGAs implementations.

REFERENCES

[1] Y. Han, I. Christoudis, K. I. Diamantaras, S. Zonouz, and A. Petropulu, "Side-channel-based code-execution monitoring systems: A survey", *IEEE Signal Processing Magazine*, vol. 36, no. 2, pp. 22–35, 2019.

[2] B. Vidal, C. Moreno, S. Fischmeister, and G. Carvajal, "Monitoring software execution flow through power consumption and dynamic time warping", *IEEE Embedded Systems Letters*, vol. 15, no. 2, pp. 101–104, 2022.

[3] J. Maillard, T. Hiscock, M. Lecomte, and C. Clavier, "Side-channel disassembly on a system-on-chip: A practical feasibility study", *Microprocessors and Microsystems*, vol. 101, p. 104 904, 2023.

[4] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "Eddie: Em-based detection of deviations in program execution", in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 333–346.

[5] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis", in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, IEEE, 2017, pp. 1–9.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", in *Advances in Cryptology — CRYPTO' 99*, Springer Berlin Heidelberg, 1999, pp. 388–397, ISBN: 978-3-540-48405-9.

[7] E. Bursztein *et al.*, "Generalized power attacks against crypto hardware using long-range deep learning", *CHES*, 2024.

[8] J. Grabinski, P. Gavrikov, J. Keuper, and M. Keuper, "Robust models are less over-confident", in *Advances in Neural Information Processing Systems*, S. Koyejo *et al.*, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 39 059–39 075.

[9] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1095–1108, ISBN: 9781450349468. DOI: 10.1145/3133956.3134081.

[10] Y. Han, M. Chan, Z. Aref, N. O. Tippenhauer, and S. Zonouz, "Hiding in plain sight? on the efficacy of power side channel-based control flow monitoring", in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 661–678.

[11] N. Lindskog, H. Englund, J. Sternby, and E. Dubrova, "Machine learning-assisted side-channel analysis for software integrity verification", in *2025 IEEE European Test Symposium (ETS)*, 2025, pp. 1–6. DOI: 10.1109/ETS63895.2025.11049653.

[12] A. D. Campos, F. Lemus-Prieto, J.-L. González-Sánchez, and A. C. Lindo, "Intrusion detection for iot environments through side-channel and machine learning techniques", *IEEE Access*, vol. 12, pp. 98 450–98 465, 2024. DOI: 10.1109/ACCESS.2024.3362670.

[13] A. Vaswani *et al.*, "Attention is all you need", in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010, ISBN: 9781510860964.

[14] A. Baevski, A. Babu, W.-N. Hsu, and M. Auli, "Efficient self-supervised learning with contextualized target representations for vision, speech and language", in *Int. Conf. on Mach. Learn.*, PMLR, 2023, pp. 1416–1429.

[15] New AE, CW305 DesignStart, [Online]. Available: https://github.com/newaetech/CW305-Arm-DesignStart (visited on 09/01/2025).

[16] J. Pallister, S. Hollis, and J. Bennett, "Beebs: Open benchmarks for energy measurements on embedded platforms", *arXiv preprint arXiv:1308.5174*, 2013.