SCANDALee: A Side-ChANnel-based DisAssembLer using Local Electromagnetic Emanations

Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar, *Fellow, IEEE*
Horst-Görtz Institute for IT Security
Ruhr University Bochum
{firstname.secondname}@rub.de

***Abstract*—Side-channel analysis has become a well-established topic in the scientific community and industry over the last one and a half decade. Somewhat surprisingly, the vast majority of work on side-channel analysis has been restricted to the "use case" of attacking cryptographic implementations through the recovery of keys. In this contribution, we show how side-channel analysis can be used for extracting code from embedded systems based on a CPU's electromagnetic emanation. There are many applications within and outside the security community where this is desirable. In cryptography, it can for example be used for recovering proprietary ciphers and security protocols. Another broad application field is general security and reverse engineering, e.g., for detecting IP violations of firmware or for debugging embedded systems when there is no debug interface or it is proprietary.**

A core feature of our approach is that we take localized electromagnetic measurements that are spatially distributed over the IC being analyzed. Given these multiple inputs, we model code extraction as a classification problem that we solve with supervised learning algorithms. We apply a variant of linear discriminant analysis to distinguish between the multiple classes. In contrast to previous approaches, which reported instruction recognition rates between 40–70%, our approach detects more than 95% of all instructions for test code, and close to 90% for real-world code. The methods are thus very relevant for use in practice. Our method performs dynamic code recognition, which has both advantages (only the program parts that are actually executed are observed) but also limitations (rare code executions are difficult to observe).

I. Introduction

Since the introduction by Kocher, Jaffe, and Jun [1] in 1996, Side-Channel Analysis (SCA) has become a serious threat for software and hardware implementations of cryptographic algorithms. Instead of focusing on mathematical properties, SCA exploits information leakage caused by the physical realization of ciphers in order to learn the secret key. SCA can be viewed as a family of passive attacks that exploits side-channel leakages like the power consumption [2] or the electro-magnetic (EM) emanation of a cryptographic device [3].

The work described in this paper has been supported in part by the German Federal Ministry of Education and Research BMBF (grant 16KIS0015, Project PhotonFX^2).

In light of the major research efforts by the scientific community and industry since the late 1990s, it is perhaps somewhat surprising that the vast majority of SCA research has been restricted to methods for extracting secret keys and developing corresponding countermeasures. There are only very few contributions considering other uses of SCA, such as reverse engineering of code from side-channel leakages. However, we believe that there are many applications for code extraction from side-channel traces. In the field of cryptography, there are (unfortunately) still many instances of practical systems where the "security" merely hinges on obscurity, i.e., weak proprietary ciphers and protocols are employed. However, especially for embedded systems, it is often difficult for security researchers to investigate the security because it is often non-trivial to overcome obscurity. An easy way of code extraction as proposed in this paper makes it considerably easier to test proprietary security solutions. A different application scenario is the detection of IP theft of embedded software, where the owner of firmware wants to test whether her code is running on a certain device. Particular in embedded systems with on-chip flash, this is often virtually impossible without major efforts. Also, outside the security context, our method has many applications, e.g., for testing software running on an embedded systems. An added advantage of our approach is that it performs dynamic code analysis, i.e., the code is observed during the actual runtime of the system. We do not claim, however, that our approach solves all problems related to embedded software testing. In particular, one can only observe code that is being executed, but not other parts of the code that are inactive.

In this paper, we build upon the relatively small number of previous publications and greatly improve the results, i.e., the instruction recognition rate. Our approach is different in two respects from previous work. First, we do not use the power consumption of the whole Microcontroller (µC), but instead exploit the localized EM emanation. By using small self-made EM probes that are placed directly above interesting parts of the hardware, e.g., the Arithmetic Logic Unit (ALU), we obtain more differentiated signals from the device under test than from power measurements. The over-all power signal of a µC has the drawback that it is the low-pass filtered sum of all circuit components, which results in a loss of information in the leakage. Second, we do not use only one position, but place the EM probe on different locations on the die of the µC. It turns out that the use of local leakage signals yields a much better rate of successfully extracted instructions.

In order to be able to compare our results, we chose the same target device as analyzed in [4], [5], the 8-bit Microchip PIC16F687 µC. This is a very widely used device and as such a good example for a simple 8-bit µC, which are used in billions of devices world-wide[1]. The PIC has an instruction set of 35 instructions and an opcode size of 14 bit. However, it seems highly likely that our results are applicable to other architectures as well.

II. Related Work

There are two related but distinct topics in the field of side-channel reverse-engineering, namely *Side-Channel Analysis for Reverse-Engineering (SCARE)* and *Side-ChANnel-based DisAssembLer (SCANDAL)*. While SCARE is mainly concerned with recovering individual values, e.g., S-box tables, of a (partially) known algorithm, SCANDAL attempts to extract code from side-channel information. In particular, by measuring and analyzing the side-channel leakage of a µC, the goal is to extract the executed instructions. In 2002, Quisquater et al. presented a correlation attack that uses a generated dictionary for each instruction, which is compared to an unknown observation [7]. Additionally, neuronal networks were used to automate the process. A recognition rate "better than 87 %" for a Complex Instruction Set Computing (CISC) processor was reported. However, the authors do not give any detailed results for a Reduced Instruction Set Computing (RISC) processor. Instead, they state that they were not able to extract the executed code on a RISC processor. In 2008, Goldack first described the analysis of an 8-bit RISC processor [4], the Microchip PIC16F687. Follow-up work was performed by Eisenbarth et al. [5]. In the latter contribution, different linear feature extraction algorithms were tested. A recognition rate of 70.1 % on test data and 40.7 % on real code was achieved. By further applying a hidden Markov model, the result on the real code could be improved to 58 %. In a recent paper, Msgna et al. report a 100 % recognition rate for a side-channel disassembler for an ATXmega163 µC [8]. While Fisher's Linear Discriminant Analysis (LDA) for the dimensionality reduction in combination with k-Nearest Neighbors (kNN) achieved a recognition rate of only 48.74 % and Means-Principal Component Analysis (PCA) with kNN a slightly better recognition rate of 56.88 %, plain PCA with kNN surprisingly outperforms all other techniques with reported results of 100 %. Even though impressive, we see several limitations to these findings. First, only 39 of the 130 instructions of the µC were supported, which limits the practical applicability. Second, the approach was not tested on real code, e.g., an implementation of the AES. As findings by others and us show, there is often quite a difference between recognition rates on test data versus real data. Finally, we tried to reproduce the approach by Msgna et al. on a different (but simpler) µC, a PIC16F687, and obtained the following results: (a) Fisher's LDA with kNN, for $k = 10$: 42.13 % and (b) PCA with kNN, for $k = 10$: 24.76 %.

[1]In 2013, approximately 20 billion 8-bit µC where shipped [6].

III. Our Contribution

This work follows a similar approach as discussed in [4], [5] and uses an identical µC for the sake of comparability. However, our approach differs significantly in the way we obtained and evaluated the side-channel leakage in order to extract the program code running on a device, leading to a very high percentage of correctly identified instructions. Our main contribution can be summarized as follows:

We are the first to present a side-channel based disassembler with a practical relevant recognition rate of 96.24 % on test data and 87.69 % on real code. In contrast, previous work only achieved at most a recognition rate of 70.1 % on test data and 40.7 % on real code [5], leaving a great amount of uncertainty.

The work in [4] and [5] was carried out with the target device running at a frequency as low as 1 MHz. As shown in [4], increasing the clock frequency to 4 MHz results in an overlap of the power consumption of individual clock cycles, assumed to decrease the success rate. We were able to confirm this assumption as described above (cf. Sect. II, Fisher's LDA). In contrast, all our results using the EM measurement were obtained using a more realistic frequency of 4 MHz. Based on the measured traces, we do not expect any significant change when increasing the clock rate even further to the maximum guaranteed frequency of 8 MHz.

We achieve this high recognition rate by evaluating the local EM emanation. The utilization of this side-channel is a novel idea in the field of extracting program code: Instead of measuring the accumulated power consumption externally, this approach enables us to include characteristics of instructions that depend on the location on the actual silicon die. Admittedly, compared to previous work, our measurement setup requires the target device to be decapsulated and thus, leaves physical traces (a fact that usually does not matter in the context of reverse engineering).

All presented results were obtained using off-the-shelf, mainly low-cost equipment and are thus easily reproducible. In fact, the total material cost (excluding the oscilloscope) is below USD 250.

IV. Side-Channel Reverse Engineering as a Classification Problem

In this paper, the code extraction problem is treated as a classical classification task. Each instruction supported by the µC corresponds to a vector $\vec{x}$, which in our case corresponds to a side-channel measurement ("trace") with l sample points. For our purposes, the term *instruction* only refers to the actual operation, e.g., `ADDLW`, `MOVF`, and so on. To limit the number of classes, operands like literals or register addresses are left out and treated as noise.

In this paper, we are in the scenario of supervised learning. Assume a µC with K instructions (classes) to be distinguished. Each instruction hence corresponds to a class $\mathcal{C}_i$, for $i \in \{1, \dots, K\}$. In the initial profiling step, we have a set of profiling traces where the correct class label

$\mathcal{C}_i$ is known for each trace. This set is called the *training set*.

The actual classification task for a trace $\vec{x}'$ with an unknown instruction now can be defined as follows: Based on the training set, generate a function $f(\vec{x}')$ that assigns the correct class $\mathcal{C}_i$ to $\vec{x}'$. This task mainly consists of two phases, the preprocessing and the actual classification.

The role of the preprocessing is to detect and emphasize particularly distinctive features that help to distinguish individual instructions. Besides, during preprocessing, parts of the trace that contain noise and are hence not necessary for the classification are discarded to reduce the computational workload. For instance, most instructions of the PIC16F687 µC need four clock cycles to be executed. Depending on the instruction, in these four clock cycles, the value of the working register is read, the computation is performed, the result is stored, and the next instruction is fetched from the instruction memory. All these sub-operations are present in the measured side-channel trace. However, most recorded data points are insignificant for the main task of classifying the executed instruction. Thus, the goal is to focus only on the relevant parts to minimize the computational effort. This is mainly done by applying a feature extraction algorithm like PCA or LDA.

The second phase is the actual classification. In this phase, traces with unknown class labels are assigned to specific classes using a feature subset and an appropriate classifier. A simple and intuitive approach of designing a classifier is based on finding similarities between the trace $\vec{x}'$ to be categorized and the training data. In [5], this is achieved by building templates for each instruction class formed by the class mean and the covariance matrix. A new trace is then assigned to the class with the highest a-posteriori probability, assuming a Gaussian probability distribution. This classification is then combined with different feature extraction algorithms mentioned above. As shown in [5], the recognition rate of 65.2% when using standard PCA can be slightly improved by Class-Mean-PCA to an average rate of 66.5% and by Fisher's LDA to 70.1%.

A different classification technique is the kNN algorithm. Instead of building templates for every class $\mathcal{C}_i$, the kNN algorithm computes the distances between a new trace and the elements of the training set. The trace is then classified depending on a majority vote of its neighbors, i.e., it is assigned to the class with the most vectors among the predefined k nearest neighbors. In practice, the kNN algorithm is also often combined with a feature extraction algorithm to decrease the computational complexity.

V. Experimental Setup

In this section, we describe our approach to acquire side-channel traces usable for the classification of the executed instructions. To this end, two problems have to be solved: *(i)* programming the target device with suitable code to obtain the training data (Sect. V-A) and *(ii)* the actual measurement of the side-channel signal, in our case the EM emanation at multiple positions (Sect. V-B).

A. Target Device

As mentioned in Sect. I, to achieve results comparable with [5], we analyzed the same target device, the 8-bit PIC16F687 µC manufactured by Microchip. The PIC16F687 uses a Harvard architecture with separate memories for instructions and data. The device employs a simple form of pipelining, where one instruction is fetched from the flash memory while the previous instruction is executed simultaneously. According to the datasheet [9], most of the 35 instructions supported by the µC are processed in four clock cycles.

A few instructions, e.g., `CALL` or `GOTO`, change the program counter. For these instructions, the subsequent instruction in the program memory is fetched in accordance with the pipelining concept. However, this prefetched instruction has to be discarded and replaced by the instruction at the position of the updated program counter, resulting in additional four clock cycles, and thus, eight clock cycles in total.

For the classification task, we took 33 of 35 instructions into account. The instructions `RETFIE` and `SLEEP` were excluded because they only are used in very specific situations and would lead to an increased complexity of the profiling step. Note that including these instructions would nevertheless be possible with appropriate handling for the arising special cases. For eight-cycle instructions, e.g., `CALL`, `RETLW`, and `RETURN`, we treated the second four cycles as a separate pseudo-instruction. For example, we consider `CALL` as a sequence of two four-cycle instructions `CALL1` and `CALL2`. Overall, we thus have 36 instructions for our classification problem, with each instruction representing one class.

Based on the approach of [5], we created a number of separate assembly files for each target instruction to be profiled. These files have the following structure: First, the µC is initialized. After that, all general purpose registers are filled with random values. The following part of the code is used for the actual profiling and testing. First, a rising edge on a pin of the µC is generated to trigger the acquisition of measurements. Then, the following three-instruction pattern is repeated until the end of the flash of the µC has been reached:

1) a preceding random instruction with random operand(s),
2) the instruction for which the assembler file was created (again with random operand(s)),
3) a subsequent random instruction with random operand(s).

Note that in our case, "random" refers to a value that is uniformly generated when creating the assembly file, but constant during the actual execution. We did not generate any randomness at runtime on the µC. Due to the limited amount of flash memory, we actually used several of such assembly files (with different random values) for each target instruction to obtain a sufficient number of samples.

B. Side-Channel Measurements

To measure the local EM emanation of the µC, we used a motorized stage to position the µC under the EM probe. This custom-made EM probe is a coil with 5 turns of 150 µm copper wire. The diameter of the coil is 0.5 mm. The low-amplitude signal picked up by the coil is amplified using two Infineon BGA427 amplifiers ICs connected in series, yielding an overall gain of more than 40 dB in the frequency range from 100 MHz to 1 GHz. This amount of amplification turned out to be sufficient for our setup and the target device.

In order to place the probe as close as possible to the die of the µC, we decapsulated the IC with white fuming nitric acid. The fragile bonding wires were protected by an epoxy compound. The side-channel measurements were recorded with a Picoscope 6403 at a sampling rate of 2.5 GS/s. The measurements were collected over an area of 1.6 mm times 1.2 mm at a spatial resolution of 0.4 mm in each direction. This resulted in 20 positions, of which in certain case only selected ones were used for the analysis. For comparison, we also recorded the current consumption over a 46 Ω shunt resistor inserted into the ground path of the µC. Note that the overall material cost of our setup is dominated by the price of approximately USD 5,000 for the oscilloscope. The remaining parts (motorized stage, EM probe, amplifiers) were made with low-cost off-the-shelf components with an overall price of less than USD 250.

The code used for the profiling (cf. Section V-A) was written to the flash of the µC using the Microchip PicKit2 programmer for each instruction. Then, the µC was reset and the acquisition of the side-channel measurements was started. Figure 1 shows an example for an EM and a power trace of the `ADDLW` instruction, including the preceding and subsequent random instructions. The power trace (Fig. 1b) exhibits a low-pass filtered shape due to the capacitance on the µC and in the measurement setup. In contrast, the EM trace (Fig. 1a) consists of a series of high-amplitude peaks that occur at the clock edges. Note that for sufficiently capturing these peaks, a high sample rate is mandatory: At a sample rate of 2.5 GS/s, the peaks have a width of approximately only 10 sample points.

VI. Results

Before starting the classification, we normalized the traces by subtracting the mean and dividing by the standard deviation. The results were generated using a training set of 2350 and a test set of 235 traces. Due to the pipelining concept (see Sect. V-A), we decided to choose a length of three instruction cycles, i.e., 12 clock cycles, for each instance. The tests were accomplished using a 10-fold cross validation: The measurements were split into ten parts where 90 % of the data is used as training data and 10 % as test data. Each experiment was conducted ten times with each part of the data being used one time as test data and the remaining nine times as subset of the training data. The results shown in the following are the average values of all ten classification runs.

To decrease the computing time, we first tried to downsample the measurement traces. However, due to this reduction much information was lost, resulting in a lower recognition rate. For this reason, we followed another approach (cf. Fig. 1a): While the peaks at the falling edge of each clock cycle seem to contain most of the information, we assumed the space in between to be irrelevant noise. Thus, as a first quick dimensionality reduction, we removed these areas of the traces. It turned out that this reduction preserves all necessary information for the forthcoming classification.

TABLE I: Recognition rates of the two classification techniques for 20 instructions. Dimensionality reduction using multiclass LDA is denoted as M-LDA, Polychotomous LDA as P-LDA. The multiclass LDA was accomplished using 34 dimensions.

Instruction	Recognition rate		Instruction	Recognition rate	
	M-LDA	P-LDA		M-LDA	P-LDA
ADDLW	70	81	IORLW	69.2	80
ADDWF	19.4	40.8	MOVLW	81.6	91.4
ANDLW	83.4	87	NOP	75.8	99.6
ANDWF	42.2	61	RLF	45	64.2
BCF	33.2	52.2	RRF	29.8	42.8
BSF	40	48	SUBLW	62.8	68.6
BTFSC	26.6	69	SUBWF	45.8	69
BTFSS	56.6	76	XORLW	55.6	74.6
CALL	78	96	XORWF	20.8	35.8
DECF	24.4	33.4	RETURN	94.4	99

A. Multiclass vs. Polychotomous LDA

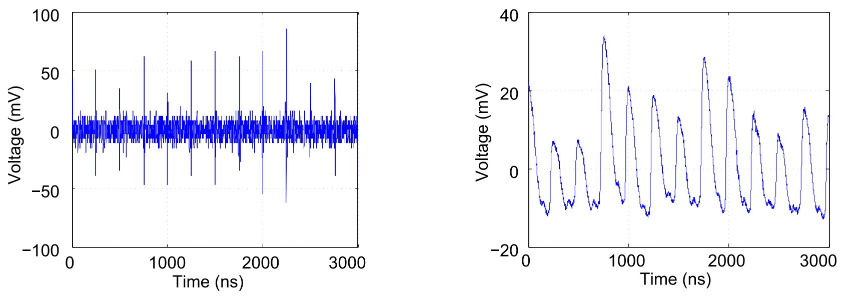
During our tests, we compared two different dimensionality reductions in combination with the kNN classifier: a multiclass LDA as a generalization of the classical 1-vs-1 LDA and a polychotomous approach proposed by Friedman in [10]. Instead of processing a new observation with one single classifier as done in the multiclass LDA, the observation is passed through every possible 1-vs-1 combination of all classes in the polychotomous approach. It is then assigned to the class with the maximum number of votes.

Our results show that multiclass LDA was clearly outperformed by the polychotomous approach. We took a measurement near the center of the chip and tested both techniques on the same training and test data. The average recognition rate was about 57 % for the multiclass LDA when using more than 13 dimensions, and 70.38 % for the 1-vs-1 approach with one dimension. Table I shows a comparison of the recognition rate of selected instructions. Due to the significantly better results of the polychotomous LDA, we decided to follow this approach in combination with the kNN classifier for the remaining analysis.

B. Combining Multiple Positions

Combining multiple sources of information meaningful is a challenging research problem. The questions that arise are for example: Should the sources be combined before or after the classification? Is it possible to find a weight vector to evaluate the votes of every individual classification or to emphasize certain sources? Also, the benefit of having several high-dimensional spaces with a large amount of precise information often yields a computationally difficult problem.

However, answering all of these questions with certainty is not necessary for the problem at hand, i.e., achieving a high instruction recognition rate. For instance, in certain



(a) EM (b) Power

Fig. 1: Side-channel trace of the `ADDLW` instruction with preceding and subsequent random instructions

TABLE II: Overview of the average recognition rates of all classes depending on the measurement position (in %)

Y	4	49.14	57.17	68.4	74.55	75.6
	3	48.81	53.87	70.36	73.79	68.26
	2	50.83	54.48	64.87	64.26	71.64
	1	62.02	65.26	60.21	65.78	69.96
		1	2	3	4	5
				X		

TABLE III: Recognition rates of the instruction `SWAPF` depending on the measurement position (in %)

Y	4	7.6	12.2	19.8	33.4	40.6
	3	7.6	9.2	29.2	39.2	18.6
	2	12	10.8	19.4	16	23
	1	19	22.6	18.8	20.4	20.8
		1	2	3	4	5
				X		

applications it might be preferable to achieve an efficient solution, but our focus is clearly on the effectiveness, i.e., the computing time becomes of secondary importance in comparison to generating results with high accuracy. We hence decided to apply a simple yet rather effective method to combine the measurements at different positions: We consecutively concatenated the traces from several positions belonging to the same class, expecting the dimensionality reduction algorithm to extract the most valuable features.

We started with computing the recognition rates of every measurement position. By taking only one position into consideration, the highest average recognition rate could be achieved at position $(x, y) = (5, 4)$, giving us a classification accuracy of 75.6 % (cf. Tab. II).

The next position was selected after the instruction with the most voting errors. For example, having chosen $(5, 4)$ as the first position, the instruction with the lowest recognition rate was `SWAPF` having a rate of 40.6 %. Table III indicates that position $(4, 3)$ gives us the second highest recognition rate for this instruction (after the one we already had chosen). By following this approach during the selection process, we could achieve the classification results given in Tab. IV.

It is striking that the highest recognition rates are given at eight positions and decrease with further additional positions. The reason could be that with the increasing number of positions, new positions were added with decreasing overall recognition rates. In our case, the ninth position was $(4, 1)$, giving us a rate of correct classifications for the instruction `SWAPF` of only 20.4 %.

TABLE IV: Recognition rates depending on the number of measurement positions

# pos.	avg. rec. rate	min. rec. rate	instruction
1	75.6 %	40.6 %	`SWAPF`
2	87.95 %	63 %	`RRF`
3	92.78 %	71.2 %	`RRF`
4	94.02 %	80.2 %	`RRF`
5	94.85 %	83.6 %	`RRF`
6	95.33 %	86.2 %	`RRF`
7	96.11 %	85.6 %	`SWAPF`
8	96.24 %	87,2 %	`SWAPF`
9	95.98 %	85.6 %	`SWAPF`
10	94.57 %	78.2 %	`SWAPF`

C. Application to Real Code

To evaluate the practical applicability of our disassembler, we ran the classification on side-channel traces acquired for real programs. Thanks go to the authors of [5], which enabled us to use the same implementation of the KeeLoq algorithm as they had used for their power analysis.

With exactly the same setup and evaluation techniques that were used for the profiling and test set, we reached a recognition rate of 77.1 % on the KeeLoq implementation, well below the rate of 96.24 % mentioned above. In [5], it is suggested that this significantly lower recognition rate is due to the distribution of the processed data (stored in the registers and the SRAM) that differs from the randomized profiling data for real code. However, for our EM measurements, we only observed a relatively minor dependency on the distribution of the processed data. Instead, we suspect that the lower recognition rate for KeeLoq is caused by the structure of the algorithm: It executes a few mainly `RLF` instructions in a short loop. Hence, not the data but the instruction distribution of our KeeLoq test code was biased towards instructions like `RLF` with a low recognition rate.

Another observation is that most of the occurrences of the instruction `ADDWF PCL` ("add register to program counter") were confused with a jump instruction, mostly `GOTO`. Considering that `GOTO` is accomplished by adding an offset to the program counter, this appears reasonable. Since `ADDWF PCL` is used frequently in the short main loop of KeeLoq, this leads to a further decrease in the recognition rate. Finally, we have to note that for our training set, we did not consider the special case that the first random instruction of the three-instruction pattern includes any type of jump instruction. However, the main loop of KeeLoq contains a relatively large number of conditional branches, causing a high error rate at successive instructions. We expect that including these types of instructions into the set of training pattern could be achieved to increase the recognition rate for this special case. Due to the special properties of KeeLoq, we decided to analyze an algorithm with a higher diversity of instructions. To this end, we used an implementation of the AES and focused on the first 550 executed instructions (after the initialization of plaintext and key). These instructions cover at least two rounds of the AES encryption and the first two rounds of the key scheduling.

We achieved a recognition rate of 84,17 % using eight positions. It turned out that a slightly higher result could be obtained by disregarding the second cycles of the jump instructions as own classes. Since the second cycles essentially all have the same function (discard the prefetched instruction, fetch a new instruction), they were often confused with each other. By considering only the first cycle, we prevent conflicting consecutive votes as, e.g., the first cycle is recognized as `CALL` and the second cycle as `RETURN2` instruction. With this technique, the final classification result was 87.69 %. In the implementation of the AES, we have again one instruction that was often confused with a jump instruction. This time, about half of the occurrences of `MOVWF PCL` was recognized as a `GOTO` instruction. `MOVWF PCL` is used for every table look-up in `SubBytes` and the key scheduling and was hence executed several times per round.

VII. Conclusion and Future Work

We presented a side-channel based disassembler using localized EM emanations. With our approach, we reached a recognition rate of 96.24 % on the test set and 87.69 % on real code (an implementation of the AES) for a Microchip PIC16F687 µC. Our novel approach significantly improves the previously reported recognition rates of 70.1 % (test set) and 40.7 % (real code), without using any further assumptions for statistical post-processing (e.g., using Markov chains as done in [5]). So far, the measurements for each position on the IC were recorded sequentially, i.e., we assume that the program flow (after a reset) is the same given constant input data. However, there are certainly situations were this assumption does not hold. In this case, each measurement position would result in a different trace (with a different sequence of instructions). In the future, we hence plan to place several EM probes on the target µC simultaneously, e.g., using small hand-wound coils or antennas printed on a circuit board.

A different aspect for future research is the extension of the presented methods to also recognize the operands of the instructions. Apart from building templates for this case—which would highly increase the efforts for creating and processing the profiling traces—classical techniques like Differential Power Analysis (DPA) may be used in cases where either the inputs or the outputs of the µC are known. To this end, including the functionality to automatically perform the necessary DPAs, once the sequence of the actual instructions has been found using the methods presented in this paper, would be of interest. Finally, the approach of localized EM analysis could also be applied to other processors, e.g., newer 32-bit μCs like the ARM Cortex-M0 (which has 57 instructions [11]) or particularly secured smartcard ICs.

Acknowledgment

The authors would like to thank Bastian Richter for the EM probe and the amplifier. Schematics are available at https://github.com/emsec/SCATools.

References

[1] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Proceedings of CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[2] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Proceedings of CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[3] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards. In *E-SMART '01: Proceedings of the International Conference on Research in Smart Cards*, pages 200–210, London, UK, 2001. Springer.

[4] Martin Goldack. Side-channel based reverse engineering for microcontrollers. Diploma thesis, Ruhr-University Bochum, 2008. https://www.emsec.rub.de/media/attachments/files/2012/10/da_goldack.pdf.

[5] Thomas Eisenbarth, Christof Paar, and Björn Weghenkel. Building a side channel based disassembler. *Transactions on Computational Science*, 10:78–99, 2010.

[6] IC Insights, Inc. MCU market on migration path to 32-bit and ARM-based devices. http://www.icinsights.com/news/bulletins/MCU-Market-On-Migration-Path-To-32bit-And-ARMbased-Devices/.

[7] Jean-Jacques Quisquater and David Samyde. Automatic code recognition for smartcards using a Kohonen neural network. In Peter Honeyman, editor, *CARDIS*. USENIX, 2002.

[8] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. Precise instruction-level side channel profiling of embedded processors. In Xinyi Huang and Jianying Zhou, editors, *Information Security Practice and Experience*, volume 8434 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2014.

[9] Microchip Technology Inc. PIC16F631/677/685/687/689/690 Data Sheet, 2007. http://ww1.microchip.com/downloads/en/DeviceDoc/41262d.pdf.

[10] Jerome H. Friedman. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University, 1996.

[11] ARM Limited. *Cortex-M0 Devices Generic User Guide*, A edition, 2009. http://infocenter.arm.com/help/topic/com.arm.doc.dui0497a/DUI0497A_cortex_m0_r0p0_generic_ug.pdf.