A Bit-Level Approach to Side Channel Based Disassembling

Valence Cristiani, Maxime Lecomte, Thomas Hiscock

A Bit-Level Approach to Side Channel Based Disassembling

Valence Cristiani[1], Maxime Lecomte[1], and Thomas Hiscock[1]

Univ. Grenoble Alpes, CEA, LETI
MINATEC Campus, F-38054 Grenoble, France
firstname.name@cea.fr

Abstract. Side-Channel Based Disassembling (SCBD) is a powerful application of side-channel analysis that allows recovering instructions executed by a processor from its physical leakages, such as the electromagnetic field (EM) emitted by the chip. These attacks directly compromise code confidentiality, but they can also reveal to an adversary many critical information on the system's internals. In this work, we propose a new approach for SCBD that directly focuses the bit encoding of an instruction using local EM leakage. We exploit a very precise bit-level leakage model and derive from it new algorithms that aim at recovering the actual bit values. We also propose strategies to automate the complex tasks of finding the best EM probe positions and combining them to improve results. On a PIC16 target, our method succeed in recovering the bits of an instruction with an average rate of 99,41% per bit. Compared to the state of the art, our disassembler is easier to train, recovers more information about instructions than just opcode and requires almost no modifications to target other processor architectures. Thus, this kind of disassemblers might become a threat to more complex processors, where side-channel disassembling has not been proved to be feasible yet.

Keywords: Side-channel analysis· Reverse engineering· Hardware security · Leakage analysis.

1 Introduction

Side-Channel Based Disassembling (SCBD) is the task of recovering instructions executed by a device based on its physical signature, known as side-channel leakages. For more than two decades many techniques and tools have been developed to extract secrets from sources such as timing variations [7], power consumption [6], electromagnetic field [15] (EM), acoustic noise [4] and many others. While side-channel analysis research is primarily concerned with the security of cryptographic primitives, there is also an active research on SCBD [3, 10, 13, 18]. Indeed an accurate quantification of the instruction leakage is a very useful security indicator for many systems. Obviously, SCBD is a direct threat to code intellectual property, which can be a requirement for manufacturers that put lots of efforts into the development of an algorithm. The instruction stream can also

reveal sensible code regions such as block ciphers or function entries. Such information can be exploited by an attacker to drive more specific attacks such as fault injections. Interestingly, SCBD can also be used as a non-intrusive malware detection mechanism [11].

The SCBD task can be regarded as a supervised machine learning classification problem, where a side-channel trace has to be associated to a sequence of executed instructions. The natural approach is to divide the global trace into instruction traces correctly labeled. These traces will then feed a learning algorithm in order to build a classifier able to make accurate predictions on the instructions that corresponds to an attack trace. However, training such a classifier is difficult in practice, as many target-specific knowledge is required to create a model. Furthermore, with complex processor architectures and deep pipelines, the switching noise generated by other activities in the core becomes preponderant. A proper randomization of all of these surrounding elements requires a huge amount of data, as well as complex profiling code snippets. Thus, while the opcode classification approach proved to work on small microcontrollers [13, 18] it is unlikely that it would scale to more complex processors.

Contributions. In this work, we propose an alternative approach to SCBD that overcome these issues. The core idea is to create a classifier directly on the bits that encode the instructions. This approach requires almost no assumptions on the target architecture, as in any processor, the bits of the instructions are transferred from memory to the processor and then manipulated, which may introduce leakages that can be exploited. Furthermore, the training is greatly simplified: the profiling can be performed on random code snippets, and by construction a bit-level approach allows having more training data available per class. We also show how exploiting very local leakages and combining EM measured at different positions greatly improves the accuracy.

Through this paper, we detail the construction of a bit-level SCBD (section 3) and evaluate its performances on a PIC16F microcontroller (section 4 and 5). We manage to get an average of 99.41% recognition rate per bit which leads to an opcode recognition rate as efficient as current state of the art. However, our disassembler recovers much more information encoded in the instructions (literal values, register numbers, etc.). These results suggest that the bit-level approach proposed is worth considering for SCBD on more complex cores.

2 Background

2.1 Structure of a side-channel disassembler

The high-level structure of a side-channel disassembler as a supervised machine learning classification task is shown in Figure 1. During a *profiling phase*, the attacker has access to a clone of the target device on which he can run arbitrary programs. He then collects side-channel data, such as the EM field, during the execution of several profiling code snippets in order to train a classifier. During

the *attack phase*, the classifier is applied on unknown traces to predict and recover instructions. An attack trace usually contains many instructions, thus a preliminary step is required to divide the trace of a program into individual instruction traces. On small devices, instructions have a constant execution time, hence a fixed-size windows is enough to extract instructions. But with complex cores this operation may require a more advanced strategy.

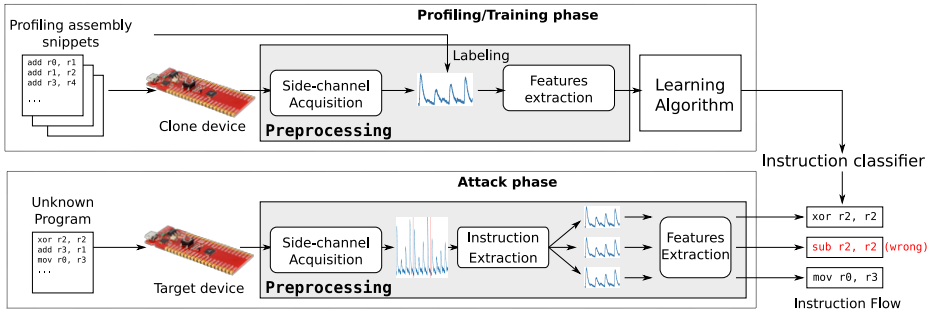


Fig. 1: High level architecture of a side-channel disassembler

The goal of a classifier is to associate a class $c \in \mathcal{C}$ to a trace $\boldsymbol{x}$ (a realization of a random variable $\boldsymbol{X}$) of p samples. In an opcode-based classifier $\mathcal{C}$ is the set of opcodes of the target architecture, for example $\mathcal{C} = \{\texttt{add}, \texttt{xor}, \texttt{load}, \dots\}$. The template attack (TA) of Chari et. al [1], used by most opcode-based classifiers [3, 13, 18] build an estimation of $\Pr(\boldsymbol{X} \mid C = c)$ during the profiling phase which is then used in the attack phase to compute $\Pr(C = c \mid \boldsymbol{X})$ thanks to Bayes' theorem. Given an unknown instruction trace $\boldsymbol{x}$, an attacker selects the class that maximizes $\Pr(C = c \mid \boldsymbol{X} = \boldsymbol{x})$. The TA models the per-class probability density $\Pr(\boldsymbol{X} \mid C = c)$ as a multivariate Gaussian distribution denoted $\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$, where $\boldsymbol{\mu}_c$ is the mean vector and $\boldsymbol{\Sigma}_c$ the covariance matrix of the distribution. For each class c, the profiling phase infers the parameters $\boldsymbol{\mu}_c$, $\boldsymbol{\Sigma}_c$ from a set of observations $(\boldsymbol{x}_c)$ using classical statistical estimators. Due to its proximity with the quadratic discriminant analysis (QDA) technique in the machine learning field, we use the terms QDA and TA interchangeably.

However, not all samples in a trace contain relevant information. A common practice is to apply feature extraction techniques to reduce the computational cost of the attack. In a nutshell, these techniques transform a trace of p samples into a trace of d samples (with $d \ll p$) while preserving –hopefully– most of the information. Both the training and the attack phases are performed in this reduced feature space. The most common feature extraction techniques include the Principal Component Analysis (PCA) and Linear Discriminant analysis (LDA).

2.2 Related work

Early side-channel based reverse engineering. The use of side-channel analysis for software reverse engineering was first suggested by Quisquarter et.

al [16] in 2002, who described an instruction classifier based on a neural network. Unfortunately, they did not provide experimental results. A leakage analysis on a Java card by Vermoen et. al [19] proved that some instructions (Java bytecodes) could be identified in a power trace, but they did not propose an instruction recovery algorithm. In his master thesis, Goldback [5] constructed a very detailed power leakage model of a 8-bit Microchip PIC16 microcontroller. He managed to perform a template attack [1] to recover up to 75% of correct instructions on a small set of 4 opcodes.

Concurrently, Novak et. al [12] and Clavier et. al [2] used side-channel analysis to perform reverse-engineering on encryption algorithms involving secret permutation tables (A3/A8). While these attacks are often referred as "Side Channel Analysis for Reverse Engineering" (SCARE), they do not allow any kind of instruction flow reconstruction.

Side-channel based disassembly. The first real side-channel based disassembler was described by Eisenbarth et. al [3] in 2010. They constructed an opcode classifier using a template attack. Thanks to Hidden Markov Models they managed to exploit prior knowledge on instruction distribution and improve the accuracy of their disassembler by a few percents. On a PIC16 microncontroller using power consumption, they managed to get a 70% recognition rate on test data and 40% on real programs. Strobel et. al [18] also described an opcode-based side-channel disassembler, working on EM. They concluded that EM contains much more information than power consumption and managed to get 90% recognition rate on real programs and 95% on test programs. Msgna et. al [10] constructed an instruction classifier based on a k-Nearest Neighbors (k-NN) algorithm. They reported a 100% instruction recognition on 35 instructions of an ATMega163 microcontroller. However, this classifier was not evaluated on real programs and [18] could not reproduce their results. Park et. al [13] constructed a SCBD on a ATMega328P exploiting knowledge of the hardware used by each instruction. They reported a 99.03% recognition rate thanks to advanced noise reduction preprocessing techniques (based on discrete wavelet transforms) and a hierarchical classification as suggested by McCann et. al [8, 9].

3 Construction of a bit-level side-channel disassembler

3.1 Challenges of bit-level instruction recovery

Processor instructions are usually encoded as a binary word (denoted $\mathcal{I}$ for the rest of this paper) of 8 up to 128 bits which contains information such as the opcode, registers and literal values used in the instructions. Our approach consists in attacking bits of $\mathcal{I}$ independently, which requires to train a distinct classifier for each bit of the instruction. Although this idea sounds pretty straightforward, it comes with viability questions that we discuss hereafter.

Distinguishing bit level variations, which impact consumption in a very tiny manner, requires a high signal to noise ratio. More than this, each bit should

have its own leakage characteristics, otherwise, distinguishing for example the 2-digits binary words `01` and `10` would not be possible. As explained in section 3.4, we suggest using EM with multiple probe positions to exploit local leakage and thus, increase chances of detecting such leakage differences between bits.

Another problem is that we suggest to attack independently some bits whose impact on the physical quantities measured are not independent. Small groups of bits of $\mathcal{I}$ are sometimes only interpretable as a whole and not separately. To address this issue, we propose to analyze only the part of the trace that corresponds to the fetching of the instruction, totally ignoring the actual execution of the instruction. While this may be interpreted as a loss of information, this drastically reduces the dependencies between the bits on the power consumption. In other words, we only analyze the update of the instruction register and not its actual execution behavior.

3.2 Leakage model and classification

A leakage model for individual bits of the instructions has to be selected in order to derive the set of classes $\mathcal{C}$ of the bit classifiers. The most common models are the Hamming Weight and Hamming distance models which at the bit level, respectively estimate the leakage as the bit value and as the bit toggle. A more accurate model known as the "signed Hamming distance" (SHD), introduced in [14] states that with precise electromagnetic measurements, the direction of the bit toggle can also be distinguished.

Based on these observations, we selected a leakage model with 3 possible target events at a given time for each bit: the bit stayed constant, it switched from 0 to 1 or switched from 1 to 0. We denote $\mathcal{T} = \{constant, 0 \rightarrow 1, 1 \rightarrow 0\}$ the set of these transitions. Following the formalism introduced in section 2.1, for each bit of the instruction a classifier with $\mathcal{C} = \mathcal{T}$ is created. It combines LDA for feature extraction and QDA for the actual classification. The training phase can be done on random instructions, with correctly labeled bit transitions, which greatly simplifies the process. Then, for an observed sequence of N instructions, the classifier is applied to all the instructions successively, and yields a finite sequence of pair: $\mathbf{T} = (t_n, p_n)_{1 \leq n \leq N}$ of bit transition associated with its probability.

3.3 From Signed Hamming distance to bit values

The sequence $\mathbf{T}$ still needs to be transformed into a sequence of bit values. However, finding the optimal bit sequence is not straightforward, as each bit prediction influences the predictions for other bits. We propose a simple algorithm that perform this task. The sequence $\mathbf{T}$ generated by the classifier is given as input to the $\texttt{FindBits}_{\texttt{Left}}$ function (shown in Algorithm 1) which maintains a state bit (b_s) and updates its value according to the transitions encountered. In case of $0 \rightarrow 1$ or $1 \rightarrow 0$, the current state bit is set to the transition final value. When no transition occurred, the current bit value is kept. The algorithm also computes a confidence in the bit value returned (p_s in Algorithm 1). This value is overwritten on a bit toggle, but is decreased when a constant transition is

taken. Intuitively, we should be less confident in a constant transition as we take the precedent value as output, which could also be wrong. We explored different strategies to update p_s in this case. Based on our experiments, an efficient one is to multiply p_s by the confidence of the actual transition. This confidence value is useful for comparing different predictions for the same bit.

Algorithm 1: `FindBits`$_{\texttt{Left}}$ (see comments for `FindBits`$_{\texttt{Right}}$)

Input: **T**, a sequence of transitions with their probability
Output: **B**, a sequence of bit values with their confidence

B $\leftarrow$ empty sequence
$b_s \leftarrow 0$
$p_s \leftarrow 0$
for $(t\,, p) \in$ **T do** `/* In a right scan: iterate in reverse order */`
  if $t \in \{0 \rightarrow 1,\ 1 \rightarrow 0\}$ **then**
    $p_s \leftarrow p$
    $b_s \leftarrow$ final value of t `/* And use the initial value of` t `instead */`
  else
    $p_s \leftarrow p_s \times p$
  Append $(p_s,\ b_s)$ to **B**
end
return B

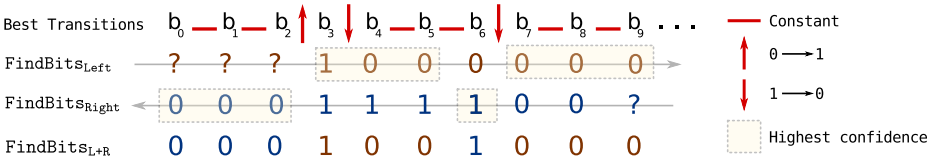


Fig. 2: Example of the different bit recovery algorithms proposed

Indeed, a transition reveals information about both the initial value and the final value. The `FindBits`$_{\texttt{Left}}$ procedure only uses the final value so far. Thus, a possible improvement is to tweak this function and define `FindBits`$_{\texttt{Right}}$ which perform the same algorithm in reverse order to exploit the information on the initial value. The right scan works the same way as the left scan but uses the initial value of the transition as the current bit value instead of the final value. The improved algorithm, denoted `FindBits`$_{L+R}$, runs both versions of `FindBits`, align the two output sequences so that the n^{th} element of each sequence corresponds to the same bit transition, and compare the two sequences selecting the bit value with the highest confidence. An example of this algorithm is shown in Figure 2. We notice that by construction `FindBits`$_{\texttt{Left}}$ and `FindBits`$_{\texttt{Right}}$ cannot give output until the first bit toggle encountered. From the previous algorithms the success rate (SR) of our classifier is defined as the number of correct bit predictions divided by the number of instructions.

3.4 Exploiting local information

Our classification uses EM field as input data rather than power consumption. Indeed, EM field with a careful probe positioning allows capturing very local effects such as single bit leakages. As we attack each bit of $\mathcal{I}$ independently, the best positions are likely to be different between the bits.

Choosing the best positions. The ideal approach to select the best probe position would be to exhaustively walk a grid of $n \times m$ positions above the circuit, run the attack on each position and select the one with the highest SR. However, this strategy can quickly become too expensive in terms of computations. For example, even a small 20×20 grid leads to 400 different positions. In our case, the longest part of the attack is the feature extraction (LDA). In order to speed up the cartography we perform the attack on a very reduced set of k points of interest (PoI). It can be viewed as an additional feature extraction step performed before the LDA during the cartography. This PoI extraction transforms the input traces $(\boldsymbol{x}_i)$ of p samples into $k \ll p$ dimensional vectors $(\boldsymbol{x}'_i)$. We choose to keep the k samples $\boldsymbol{x}(t)$ that maximize the Mean Difference (MD):

$$MD(t) = \sum_{\substack{c_1, c_2 \in \mathcal{T}, \\ c_1 \neq c_2}} \Big| \mathrm{E}[\boldsymbol{X}(t) \mid C = c_1] - \mathrm{E}[\boldsymbol{X}(t) \mid C = c_2] \Big|$$

Choosing the value of k can be done empirically observing the evolution of the SR as k decreases, at some fixed positions. Although the SRs may be lower with this additional step, we assume that this should not drastically change the ranking of the positions. Once the best positions has been found, the actual attack can be run with either a higher value of k or without the PoI extraction.

Algorithm 2: `FindPos`

Input: $\mathbf{M}$, a list of EM measurements at different positions in a set $\mathcal{G}$
Output: $\mathcal{P} \subset \mathcal{G}$, a subset of positions to be combined

`/* The` $SR(\mathcal{P}, \mathbf{M})$ `function concatenates EM data at positions` $\mathcal{P}$`, runs the attack, computes and returns the` SR `*/`
$\mathcal{P} \leftarrow \{\}$
for $step \leftarrow 0$ **to** $step_{max}$ **do**
  $best \leftarrow \arg\max_{pos \in \mathcal{G} \setminus \mathcal{P}} SR(\mathcal{P} \cup \{pos\}, \mathbf{M})$
  if $|SR(\mathcal{P}, \mathbf{M}) - SR(\mathcal{P} \cup \{best\}, \mathbf{M})| < \epsilon$ **then return** $\mathcal{P}$
  $\mathcal{P} \leftarrow \mathcal{P} \cup best$
end
return $\mathcal{P}$

Combining different positions. In a previous work, Strobel et. al. [18] combined EM traces acquired from different positions by concatenating them before the features extraction. However, finding the best combination of positions

(which maximise the SR for instance) is hard in practice due to an exponential-size search space. We propose a simple greedy algorithm (see algorithm 2) that searches a good subset of positions to attack one bit. In a nutshell, the algorithm builds iteratively a set $\mathcal{P}$ of best positions. At each iteration, the algorithm attempts to add each one of the remaining positions to $\mathcal{P}$. The one that improves the most the SR is added to $\mathcal{P}$. The algorithm may exit earlier if the success rate does not improve enough (this threshold is defined by ϵ).

4 Leakage analysis of the PIC16F

4.1 Overview of the PIC16F

Our experiments are conducted on a PIC16F15376 Microchip microcontroller from the MPLAB Xpress evaluation board. Besides being ubiquitous in embedded systems, this family of PIC microcontrollers is a common reference in the SCBD literature [3, 5, 18] and allows a fair comparison of our results. The PIC16F15376 has around 50 instructions which are encoded on 14 bits. A typical instruction contains from 3 to 6 bits that are used to match the instruction (the opcode in some sense). The remaining bits encode the arguments of the instruction such as a control bits, source/destination registers or literal values.

Architecture. A simplified internal architecture of the PIC16F is depicted on Figure 3. Excluding jumps, all instructions require 4 clock cycles to complete. During the execution of an instruction, the next one is prefetched from FLASH memory, thus this processor has a 2-stage pipeline: prefetch and execution. The instruction bit leakages are most likely to be caused by this prefetching and the instruction register activity. Jump instructions require 4 additional clock cycles to be executed but can be regarded as the actual instruction followed by a `nop`. This dummy instruction is actually only used to refill the pipeline after the jump and our disassembler always detect it as a `nop`.

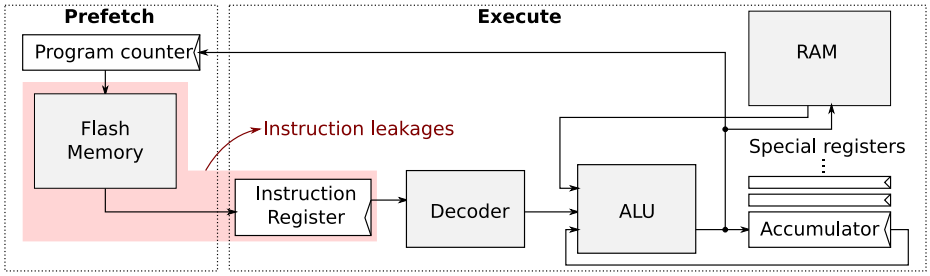


Fig. 3: Architecture of the PIC16F

Side-channel behavior. A taste of EM and power side-channel traces obtained during the execution of 3 instructions are shown in Figure 4. Both were acquired

using the measurement setup described in section 4.2. We stress that in all of the experiments, the PIC16 is clocked at 20 Mhz, except for Figure 4 where the clock was reduced to 1MHz to distinguish power peaks (at 20Mhz, the power curve is flat). The 4 execution cycles are clearly visible on the traces. As expected, the EM behavior is much more local: some peaks have different amplitudes and small temporal shifts based on the probe position.

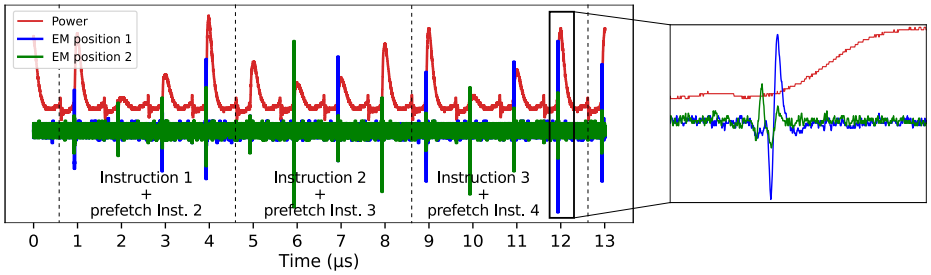


Fig. 4: Power and EM field measured during the execution of 3 instructions.

4.2 Our experimental setup

Our experimental setup is presented in Figure 5. We acquire the near field electromagnetic emanations of the PIC16F through an EM Langer probe, an ICR HH 100 27 with a bandwidth of 6 GHz. The probe is placed over the IC package without any depackaging, at less than $500\mu m$ from the package thanks to a high precision motorized XYZ table. The probe is connected through a low-noise amplifier to a digital oscilloscope (DSO) from Rohde & Schwarz (RTO 2024) which has a bandwidth of 2 GHz and a sample rate set to 10 GS/s. The PIC is clocked by an external reference set at 20 MHz. A GPIO of the PIC is used to synchronize the oscilloscope acquisition with the computation.

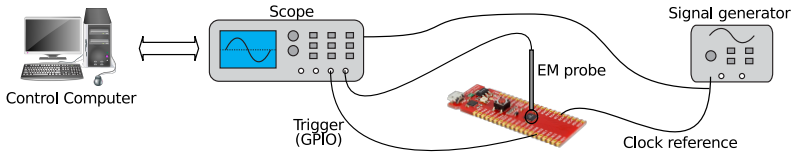


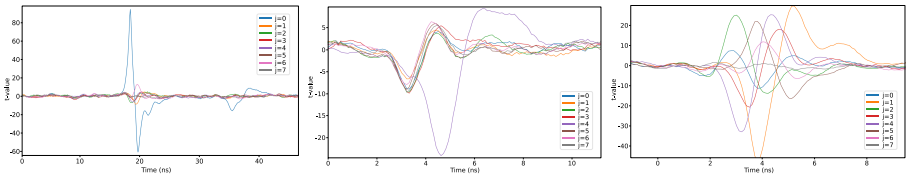
Fig. 5: Experimental setup

With these settings, the 4 clock cycles of an instruction last for 200 ns and represents 2000 samples of the oscilloscope. A typical trace may span over several milliseconds and is made of thousands of instructions. To avoid synchronization issues, the clock of the PIC is generated by a signal generator (FI5350GA) which is also connected to the reference clock of the DSO. This setup ensures that the PIC and the DSO stay synchronized and that no post processing is required to divide a trace into individual instructions.

4.3 Study of single bit leakages

This section presents three experimental results which confirm that there exists probe positions where 1) single bit leakages actually occur, 2) each bit influences the EM field independently from other bits and 3) the SHD leakage model suggested in section 3.2 is appropriate. These experiments are seen as pre-attack analysis to validate the requirements for a bit-level disassembler to be successful. For concision and simplicity, only the behavior of the 8 lower bits of $\mathcal{I}$ (14-bit PIC instructions) are analyzed. These specific bits encode literal values and can be set to an arbitrary value still creating a valid instruction if the remaining 6 upper bits of $\mathcal{I}$ are set to an opcode that uses a literal value. As an example, we will use `movlw k` (which is encoded as $\texttt{110000}\|k_{\text{base 2}}$) that loads the literal value `k` into the processor accumulator register.

Leakage differences between bits. To demonstrate that single bit leakages are distinguishable, we perform Welch's t-test [17] between traces of `movlw 0` and `movlw` 2^j, with $0 \leq j \leq 7$. This test evaluates whether there are significant differences on traces when a single bit of the instruction changes. In the profiling code, a `nop` (encoded with fourteen 0s) is placed before each `movlw` instruction, so that the test works for any leakage model. The t-test is performed on all 400 probe positions of a square ($2mm \times 2mm$) grid (20×20) centered over the chip. Many probe positions with a successful t-tests (that goes over a threshold of 4.5) were found, which means that the SNR in our setup is good enough to detect single bit variations. Figure 6a and 6b show respectively positions where a t-test for $j = 0$ and for $j = 4$ are very different from the others. Intuitively, these two positions may bring useful information to determine respectively the value of bit 0 and bit 4 of $\mathcal{I}$. Figure 6c shows a position where all t-tests are distinct from each other. The same experiment was performed with other literal instructions such as `addlw` or `xor` and gave similar results.



(a) Good position to distinguish bit 0

(b) Good position to distinguish bit 4

(c) Position where t-tests of all bits are distinct

Fig. 6: Single bit T-Tests results at different positions

Leakage independence of bits. The second experiment aims at verifying that each bit of $\mathcal{I}$ contributes independently to the EM field. We use the notation $\mathcal{L}(\texttt{movlw k})$ to denote the measured EM field during the prefetching of the instruction `movlw k` (as in the previous experiment, a `nop` is prepended to

each instruction). In our setup, $\mathcal{L}(\cdot)$ returns a 2000-dimensional vector, as the prefetching last for one instruction. To model the leakage strictly caused by an 8 bits literal value we define the leakage function $\mathcal{L}_{literal}$ simply by:

$$\mathcal{L}_{literal}(\texttt{k}) = \mathcal{L}(\texttt{movlw k}) - \mathcal{L}(\texttt{movlw 0})$$

If the leakage of bits are independent, one would expect that the leakages of individual bits can be summed to obtain the leakage of a given word, formally for any subset J of $\{0, 1, 2, 3, 4, 5, 6, 7\}$, one should have:

$$\sum_{j \in J} \mathcal{L}_{literal}(2^j) = \mathcal{L}_{literal}\left(\sum_{j \in J} 2^j\right)$$

We verified this equation empirically for some of the probe positions found in the previous experiments. Figure 7 shows an example where $J = \{0, 1, 2, 3, 4, 5, 6, 7\}$. All the small amplitude curves represent the leakage function $\mathcal{L}_{literal}(2^j)$ for $0 \leq j \leq 7$. The dark and light blue lines represent respectively the sum of all the individual leakages and the leakage of 255 which is equal to $\sum_{j=0}^{7} 2^j$. These two lines clearly seem to match. One could argue that this experiment is not enough to really show that each bit contributes independently to the global leakage. However, it still increases our confidence in the feasibility of the attack.

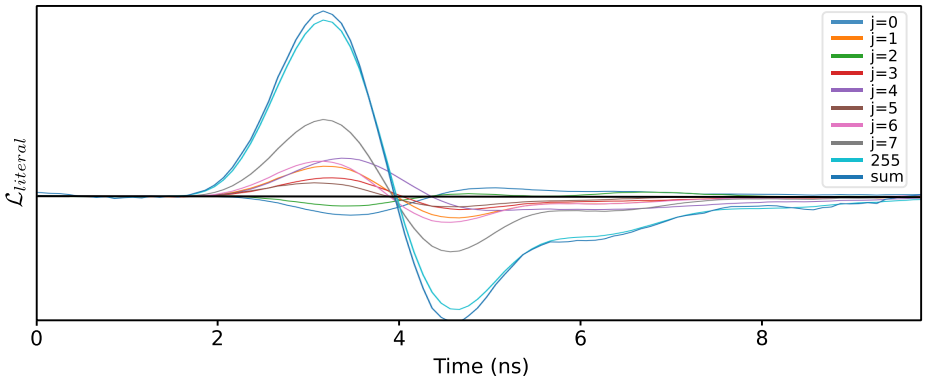
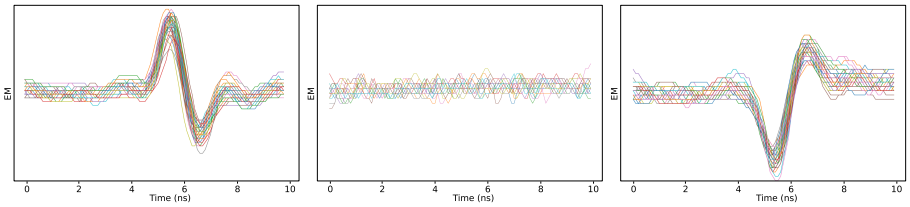


Fig. 7: Leakage independence

Leakage model. All experiments presented so far analyzed the fetching of instructions preceded by a `nop`, so that the results are agnostic to the leakage model. To confirm that the signed Hamming distance is an appropriate model, we choose some of the probe positions with high t-test associated to one particular bit of $\mathcal{I}$. Then, we analyzed the leakages in terms of transitions of this bit (all the other bits being constant). Figure 8 shows the EM traces for multiple transitions of the bit 0. This clearly illustrates that the three classes from the SHD model create a good partitioning of traces.



(a) $0 \rightarrow 1$ (b) $0 \rightarrow 0$ or $1 \rightarrow 1$ (c) $1 \rightarrow 0$

Fig. 8: EM traces grouped according to the transition of bit 0

5 Evaluation

This section presents the results of our bit-level SCBD. The training and evaluation phases use two different sets made of 2000 random valid instructions. Each acquisition is averaged 1000 times to improve the SNR. The results were also confirmed on simple programs written in C. The disassembler first applies a PoI extraction of $k = 50$ points that maximize the MD, then applies a LDA to the results and keeps 2 components. Then, the fourteen QDA-based classifiers, one for each bit of $\mathcal{I}$, are applied to recover the 1999 bit transitions among the 3 transition classes $\mathcal{T}$ introduced in section 3.2. Finally, the algorithms described in section 3.3 are applied to recover the 1999 corresponding bit values.

5.1 Mono-spatial attack

The attack was first conducted for each bit on each of the 20×20 grid (400 positions) that was used for our leakage analysis. The SR of the attack (using the $\texttt{FindBits}_{L+R}$ algorithm) on all the grid and for each bit is shown in Figure 9 (a Gaussian interpolation has been applied to the raw data). Surprisingly, each bit has its own spatial signature: the "hot areas", where the attack has a better success rate, strongly depend on the bit. The best success rates obtained for each bit are given in table 1. The $\texttt{FindBits}_{L+R}$ algorithm slightly improves the accuracy of the attack. While most bits are recovered with a high accuracy, a few (bits 8, 9, 10, 11), hardly get above 80%. These results can be improved by combining measurements from multiple positions.

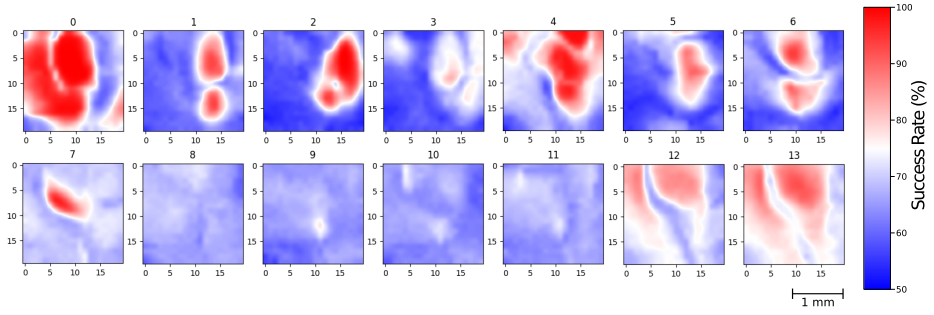


Fig. 9: Cartography of the SR of the bit-level classifiers

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$\texttt{FindBits}_{\texttt{Left}}$	100	94.1	99.2	83.1	97.0	90.7	93.5	97.4	71.8	78.6	75.9	72.8	88.7	92.0
$\texttt{FindBits}_{L+R}$	100	94.8	99.5	85.7	98.2	93.1	94.6	97.9	74.5	80.6	78.7	74.2	90.9	93.5

Table 1: Success rate at the best probe position for each bit

5.2 Multi-spatial attack

We evaluate the multi-position attack described in section 3.4 by collecting and combining data from up to 14 positions using algorithm 2. Figure 10 shows how adding more positions affects the SR of each bit. Note that for a given bit, we stop collecting new positions when the SR improvement is too low. These results show that the low SR of some bits in the mono-spatial case (bits 8 to 11) can be brought up to 97 % and more with several additional positions. Once the best position combination has been found it is still possible to increase the number of sample kept by the PoI extraction: table 2 shows the SR of a multi-spatial attack where the number of PoI is higher ($k = 400$). All the SR are above 98.4 %, we achieved a 100 % SR for 6 bits. The average of the 14 SR is 99.41% which leads to 95% of the instructions being recovered without any faults. The acquisition time for this attack is about one hour, the training of all the classifiers takes approximately 30 minutes and the actual attack is instantaneous.

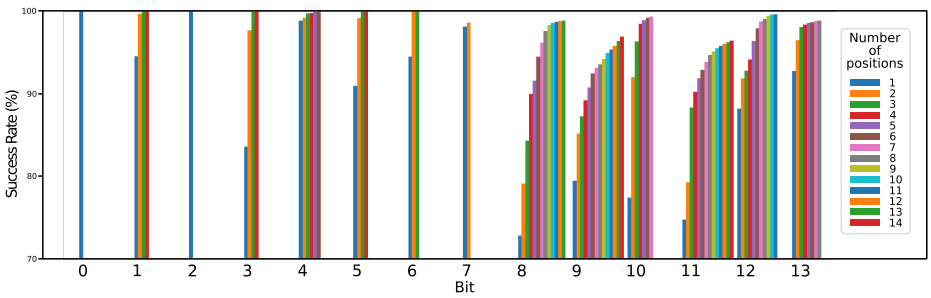


Fig. 10: Evolution of the success rate by adding new positions

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Success rate	100	100	100	100	99.8	100	100	98.5	98.6	98.9	99.3	98.4	99.6	98.7
Used positions	1	4	1	4	6	4	3	2	13	14	7	14	11	8

Table 2: Success rate for a multi-spatial attack

5.3 Template portability

In a realistic context, the training phase would be performed on a clone of the target. This introduces the risk of overfitting on the clone device characteristics. Our first attempt to port the attack was no exception to the rule: applying our classifier to a different target completely failed. However, the SR cartography for two different targets shown in figure 9 reveal clear similarities, which suggests

that the attacks behave almost the same on the two targets. More precisely the SR cartography is almost the same but shifted by a constant vector of norm around $300\mu m$. We successfully conducted an attack between the two target with roughly the same SR as in the mono-target case simply by shifting all the probe position at the acquisition time on the second circuit. In a real attack scenario, we argue that the shift vector could be brute-forced (until a high SR is reached) by attacking a known sequence of instructions such as the boot code.

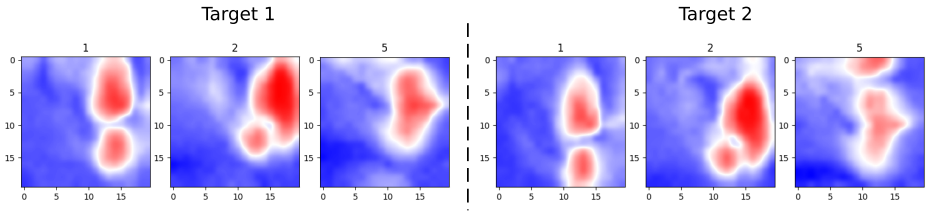


Fig. 11: Leakage cartography for two different devices (same scale as Figure 9)

6 Conclusion and further work

In this work, we described a new kind of side-channel disassembler that uses bit-level classifiers to recover instructions from non-invasive EM measurements. This approach requires a very precise experimental setup to discriminate small bits variations in traces, especially on very a low-power device like a PIC16F microcontroller. Fortunately, the algorithms proposed in this paper can fully automate the recovering process. Furthermore, we observed that the disassembler is portable between different chips of the same family, which makes this kind of attacks truly realistic. A bit-level instruction disassembler is a huge gain in terms of genericity. The training process is greatly simplified compared to an opcode classifier because it can be performed on random binaries instead of carefully crafted assembly snippets. We demonstrated that such a disassembler achieve good recognition rate, with an average success rate of 99.41% on a bit level and 95% on the full 14-bits instruction. This result may be improved by exploiting prior knowledge on the program such as instruction transition probability, invalid opcode, etc...

It seems that our approach can be extended to recover other valuable information from processors such as runtime register values. Moreover, this work opens interesting perspectives regarding the side-channel disassembling on pipelined processors, which remains an open problem. Future work will aim at validating our approach on more complex processors.

Acknowledgments

The authors would like to thanks the reviewers for their helpful comments. This work was funded thanks to the French national program "Programme d'Investissement d'Avenir IRT Nanoelec" ANR-10-AIRT-05.

References

1. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems (2002)
2. Clavier, C.: Side channel analysis for reverse engineering (SCARE), an improved attack against a secret A3/A8 GSM algorithm (2004)
3. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. In: Transactions on computational science (2010)
4. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Annual Cryptology Conference (2014)
5. Goldack, M., Paar, I.C.: Side-channel based reverse engineering for microcontrollers. Master's thesis, Ruhr-Universität Bochum, Germany (2008)
6. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference (1999)
7. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In: Annual International Cryptology Conference (1996)
8. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In: USENIX Security Symposium (2017)
9. McCann, D., Whitnall, C., Oswald, E.: ELMO: Emulating leaks for the ARM Cortex-M0 without access to a side channel lab. IACR Cryptology ePrint Archive (2016)
10. Msgna, M., Markantonakis, K., Mayes, K.: Precise instruction-level side channel profiling of embedded processors. In: International Conference on Information Security Practice and Experience (2014)
11. Msgna, M., Markantonakis, K., Naccache, D., Mayes, K.: Verifying software integrity in embedded systems: A side channel approach. In: International Workshop on Constructive Side-Channel Analysis and Secure Design (2014)
12. Novak, R.: Side-channel based reverse engineering of secret algorithms. In: Proceedings of the Electrotechnical and Computer Science Conference (2003)
13. Park, J., Xu, X., Jin, Y., Forte, D., Tehranipoor, M.: Power-based side-channel instruction-level disassembler. In: Design Automation Conference (2018)
14. Peeters, E., Standaert, F.X., Quisquater, J.J.: Power and electromagnetic analysis: Improved model, consequences and comparisons. The VLSI journal (2007)
15. Quisquater, J.J., Samyde, D.: Electromagnetic analysis: Measures and countermeasures for smart cards. In: International Conference on Research in Smart Cards (2001)
16. Quisquater, J.J., Samyde, D.: Automatic code recognition for smart cards using a kohonen neural network. In: Proceedings of the Smart Card Research and Advanced Application Conference (2002)
17. Schneider, T., Moradi, A.: Leakage assessment methodology. In: International Workshop on Cryptographic Hardware and Embedded Systems (2015)
18. Strobel, D., Bache, F., Oswald, D., Schellenberg, F., Paar, C.: SCANDALee: a side-channel-based disassembler using local electromagnetic emanations. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (2015)
19. Vermoen, D., Witteman, M., Gaydadjiev, G.N.: Reverse engineering java card applets using power analysis. In: International Workshop on Information Security Theory and Practices (2007)