A Hierarchical Classification Method for High-Accuracy Instruction Disassembly with Near-Field EM Measurements

VISHNUVARDHAN V. IYER *, ADITYA THIMMAIAH, MICHAEL ORSHANSKY, ANDREAS GERSTLAUER, and ALI E. YILMAZ

The University of Texas at Austin, Texas, USA

Electromagnetic (EM) fields have been extensively studied as potent side-channel tools for testing the security of hardware implementations. In this work, a low-cost side-channel disassembler that uses fine-grained EM signals to predict a program's execution trace with high accuracy is proposed. Unlike conventional side-channel disassemblers, the proposed disassembler does not require extensive randomized instantiations of instructions to profile them, instead relying on leakage-model-informed sub-sampling of potential architectural states resulting from instruction execution, which is further augmented by using a structured hierarchical approach. The proposed disassembler consists of two phases: (i) In the feature-selection phase, signals are collected with a relatively small EM probe, performing high-resolution scans near the chip surface, as profiling codes are executed. The measured signals from the numerous probe configurations are compiled into a hierarchical database by storing the min-max envelopes of the probed EM fields and differential signals derived from them, a novel dimension that increases the potency of the analysis. The envelope-to-envelope distances are evaluated throughout the hierarchy to identify optimal measurement configurations that maximize the distance between each pair of instruction classes. (ii) In the classification phase, signals measured for unknown instructions using optimal measurement configurations identified in the first phase are compared to the envelopes stored in the database to perform binary classification with majority voting, identifying candidate instruction classes at each hierarchical stage. Both phases of the disassembler rely on a 4-stage hierarchical grouping of instructions by their length, size, operands, and functions. The proposed disassembler is shown to recover ~97-99% of instructions from several test and application benchmark programs executed on the AT89S51 microcontroller.

CCS CONCEPTS • **Security and privacy→ Security in hardware →Hardware attacks and countermeasures → Side-channel analysis and countermeasures**

Additional Keywords and Phrases: Electromagnetic side-channel, embedded processors, instruction-level disassembly, side-channel security

1 INTRODUCTION

On-chip computations impact the electromagnetic (EM) fields emanated as well as the power consumed by embedded systems [1]-[10], causing information about the operations they execute to leak through these side channels. By probing these fields and exploiting variations in the measured signals, side-channel analysis (SCA) attacks can non-invasively recover information about target processes even in embedded processors that execute general-purpose programs. At the highest fidelity, EM SCA can potentially disassemble a program's execution trace from a device under test (DUT) at the

This work was supported in part by NSF Grant CCF-1901446.
Author addresses: Vishnuvardhan V. Iyer, Aditya Thimmaiah, Michael Orshansky, Andreas Gerstlauer, and Ali E. Yilmaz are with the University of Texas at Austin, Austin, TX 78712. Email: {vishnuv.iyer, auditt, orshansky, gerstl, ayilmaz}@utexas.edu

instruction level. Although such instruction-level disassemblers based on power SCA are well documented [3]-[5], only a few attempts based on EM SCA are reported in the literature [6]-[8].

Disassemblers using relatively large EM [6], or power [3]-[5] probes aggregate the fields emanated or power consumed by many/all system components throughout the DUT. Thus, any potential features in the measured signals that can distinguish instructions are heavily obfuscated by algorithmic noise from uncorrelated processes in addition to measurement noise from the environment and the sensor setup [10]. Such coarse-grained EM/power SCA setups generally require extensive measurements to quantify and filter out noise [3]-[6]. Contrarily, fine-grained EM SCA setups [7],[9],[10], which use relatively small probes, are sensitive to the fields emanated by a subset of system components near the probes because EM emanations decay rapidly with distance and are polarized. Indeed, when probes are *appropriately positioned and oriented*, fine-grained EM SCA can improve the success rate of disassembly [7]. Thus, fine-grained EM SCA attacks first scan for effective measurement configurations that have high signal-to-noise ratios and then use these low-noise configurations to actually extract information [7], [9]. However, the "acquisition cost" of finding optimal configurations in existing fine-grained approaches can be prohibitively large [10]. The efficiency of a disassembler directly relates to how well the instructions are profiled during the initial acquisition phase, which dictates the acquisition cost in terms of measurement time and storage requirements. A naïve profiling approach involves instantiating each instruction with all possible combinations of different operands, addresses, and data present in architectural registers, such as program counters, stack, etc. [3]-[6]. To feasibly profile instructions, conventional SCA-based disassemblers typically sub-sample this space of architectural states by randomly instantiating instructions several times with different operand values and machine states. This approach has limited feasibility for fine-grained EM SCA-based disassemblers because of the high acquisition cost of searching a 5-D space of potential optimal measurement configurations— the possible probe locations (3-D), orientations (1-D), and observation times (1-D) —as the DUT executes many instantiations of each instruction [10]; e.g., the setup used in this article would require $\sim 5000\times$ more signals to be collected compared to using a single probe configuration. The scalability of such methods further reduces as the size of the instruction set N increases. Indeed, fine-grained EM SCA approaches using the random instantiations method for profiling instructions [7] have been limited to small instruction sets. Random instantiations may also miss critical corner cases which can lead to potential misclassifications in the classification phase.

In this paper, a novel scalable and effective instruction disassembler using fine-grained EM signals is proposed. As in previous SCA-based disassemblers [3]-[7], the proposed method has 2 phases. *The feature-selection phase* identifies optimal measurement configurations and corresponding signal features. After this phase, *the classification phase* identifies instructions from signals measured as the DUT executes an arbitrary code. It collects signals using only the selected set of configurations and evaluates them according to the features identified in the first phase. To support large instruction sets, the disassembly is performed hierarchically; a 4-stage hierarchy—consisting of an instruction's cycle length, size, operands used, and functions implemented (Fig. 1)—is used; and the feature-selection phase is performed bottom-up, while the classification phase is performed top-down through the hierarchy. A hierarchical classification allows evaluators to identify distinct leakage-mode informed features pertinent to each stage. Furthermore, ensuring high classification success rate in upper hierarchical levels enables evaluators to still recover key information about the executed instructions even if accuracy in separating details on lower levels is reduced.

The hierarchical classification is combined with a leakage model-informed sub-sampling of potential architectural states to profile instructions and identify optimal features for each stage in a feasible and scalable manner. The feature-selection phase uses a Hamming weight (HW) leakage model to design "profiling codes" consisting of a condensed set of test instructions such that—if there was no noise and if the leakage model was valid—the signals measured as the DUT

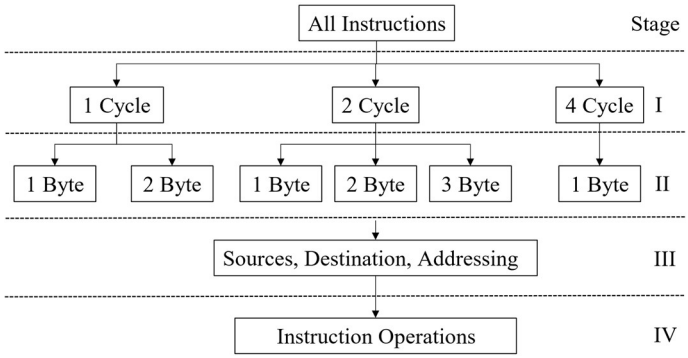


Fig.1. Hierarchical grouping of instructions based on length (I), size (II), operands (III), and functions (IV).

executes these codes would min-max bound the signals that would be measured as the DUT executes all possible instantiations of the profiled instructions. The min-max signal envelopes for each instruction class are collected and stored in the hierarchical database, as the profiling codes are executed. Configurations where pairs of instruction classes can most easily be separated are identified. The signals measured at these configurations are the "features" that are used to classify instructions using binary classification with majority voting [5] in the next phase.

In addition to measured signals, this work also uses novel "differential signals" derived from them to improve success rates. These signals capture the impact of an instruction on the architectural state over multiple cycles. The capabilities of the disassembler are further augmented by assuming branches taken and not-taken as separate instruction classes, enabling control-flow prediction. The proposed method enables high-resolution measurements at a low acquisition cost, efficiently identifying highly potent features within a large search space. As a result of the leakage-model-informed feature selection, and hierarchical classification, improved success rates are observed for application benchmarks, compared to alternative methods [4], [7].

The contributions of this work can be summarized as follows:

- Fine-grained EM SCA-based disassembly is performed by identifying optimal probe configurations and corresponding signal envelopes during the feature-selection phase.
- In addition to directly probed signals, novel differential signals derived from them are used as features.
- Control-flow leakage prediction is enabled with input-constrained analysis of branch instructions.
- Success rates of ~99% and ~97% are observed when the proposed method is used to disassemble test codes and application benchmarks from the Dalton project [14] executed by a AT89S51 microcontroller unit implementing the i8051 instruction set [12] ($N = 90$ instructions).

The rest of the paper is organized as follows: Section 2 compares various disassemblers with the proposed approach. Section 3 presents relevant background for the proposed experiments. Section 4 details the feature-selection method. Section 5 describes the classification method. Section 6 presents the measurement results. Section 7 concludes the work.

2 OVERVIEW

This section reviews previous SCA-based disassemblers and presents an overview of the proposed approach.

Table 1: Comparison of Relevant Work

	[4]	[6]	[7]	[5]	[8]	[27]	**This Work**
DUT	PIC16F 687	ATMega 328	PIC 16F687	ATMega 328P	PIC16F15376	Cortex M0	AT89S51
# of Instr. (N)	33	2	33	112	50	17	90
Side-Channel	Power	Coarse-grained EM	Fine-grained EM	Power	Fine-grained EM	Power	Fine-grained EM
# of Samples Measured per Instr. ($N_{\mathrm{pc}} \times N_{\mathrm{t}} \times \overline{N}_{\mathrm{inst}}$)	$\sim 2 \times 10^6$ ($1 \times 1000 \times 2000$)	$\sim 2 \times 10^4$ ($1 \times 100 \times 200$)	$\sim 1.2 \times 10^8$ ($20 \times 2500 \times 2350$)	$\sim 1.5 \times 10^5$ ($1 \times 50 \times 3000$)	$\sim 3.2 \times 10^7$ ($400 \times 2000 \times 40$)	$\sim 1.1 \times 10^7$ ($1 \times 6000 \times 1768$)	$\sim 4.7 \times 10^7$ ($5200 \times 1000 \times 9$)
Success (test code)	~70.1%	100%	~96.2%	~99.0%	~95.0%	~99.0%	~99.3%
Success (application code)	~50.8%	–	~87.7%	–	–	~88.2%	~97.3%

2.1 Related Work

Various SCA-based methods exist for recovering information about target processes on embedded systems. Code-monitoring with SCA is most often used to identify fixed instruction sequences, separate basic blocks, and predict control flow [1],[2] based on some *a priori* knowledge of an evaluated benchmark. Using SCA to disassemble individual instructions from an arbitrary unknown code as in [4]-[8] is far more challenging in part because each instruction impacts a multitude of architectural blocks differently. Disassemblers can be compared based on their success rates and their acquisition cost. While success rate is simply the ratio of correctly identified instructions and total number of executed instructions, the acquisition cost is a function of the number of sensor configurations used during profiling N_{pc}, the number of instantiations performed to characterize each instruction $\overline{N}_{\mathrm{inst}}$, and the number of samples collected for each of these measurements N_t. The acquisition cost in this work only accounts for samples stored post measurement collection, and does not quantify repeated measurements and averaging performed by the oscilloscope software.[1]

Instruction disassembly based on coarse-grained EM or power SCA setups [4]-[6] uses a single sensor configuration ($N_{\mathrm{pc}} = 1$) and requires significant post-processing of the signals measured as the DUT executes an extensive set of test instructions. In [4], a power SCA-based disassembler, using principal component analysis (PCA) for feature selection and a multivariate Gaussian classifier, was proposed to evaluate a small instruction set ($N = 33$). It correctly recognized ~71% and ~51% of instructions in test code and application benchmarks, respectively. The method in [4] assumes some *a priori* knowledge of the code, however, as it applies hidden Markov models to blocks of the executed code. In [6], a coarse-grained EM SCA-based disassembler, using PCA with frequency-domain signals for feature selection and AdaBoost, support vector machine, and other methods for classification, was proposed. It was able to distinguish 2 instructions with

[1] Please note that the acquisition cost here only quantifies storage requirements and not acquisition time. Acquisition time is related to several setup-dependent factors including oscilloscope features, DUT parameters, averaging method, etc., some of which are not always available in literature.

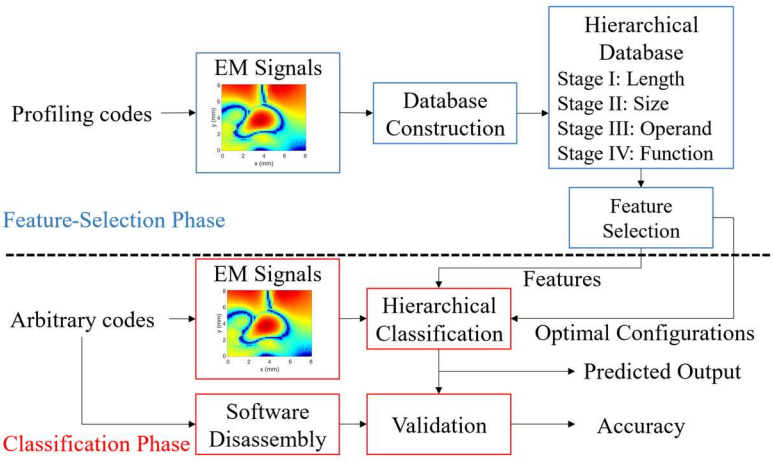


Fig.2. Overview of the proposed approach.

a 100% success rate. Unfortunately, the method's performance for the remaining instructions was not evaluated in [6]. A larger instruction set ($N > 100$) was evaluated in [5] with a power SCA-based disassembler, using Kullback-Leibler (KL) divergence for feature selection and quadratic discriminant analysis for classification. The method disassembled a test code with ~99% success rate. Although [5] used hierarchical classification, included an extra method to improve success rates for application benchmarks, and recovered 2 instructions implemented in one such code with 92% success rate, the method was not evaluated comprehensively on real-world application benchmarks. In [27], an instruction disassembler targeting a Cortex M0 processor was proposed, implementing KL divergence for feature selection and classification algorithms demonstrated in [5], which was further enhanced by using models based on multi-layer perceptron and convolutional neural network. While the method recognized ~99% and ~88% of instructions in test code and application benchmarks respectively, the disassembly was limited to a small subset of the full instruction set ($N = 17$).

Instruction disassembly based on fine-grained EM SCA was demonstrated in [7],[8]. A small instruction set ($N = 33$) was evaluated in [7] using linear discriminant analysis for feature selection and a k-Nearest Neighbor algorithm for classification. While the disassembler recognized ~96% of the instructions in a test code and ~88% of them in application benchmarks, the approach in [7] is an invasive method that requires decapsulation of the DUT to constrain the search space of configurations during feature selection. A similar fine-grained setup in [8] targeted a slightly larger instruction set ($N = 50$) by performing bit-level disassembly of opcodes, training quadrature discriminant analysis-based classifiers to identify individual bit transitions as instructions are pre-fetched. Although the disassembler recognized 95% of instructions in test codes, it was not evaluated on real benchmarks.

While the methods proposed in [4]-[8], [27] (Table 1) have very high success rates when disassembling test codes that follow the same structure/template as the profiling codes they use to select features, their success rates either decrease markedly or are unknown when disassembling application benchmarks; moreover, the methods in [4],[6],[7], [27] which were developed and tested with only limited number of instructions, may not scale well as N, the instruction set's size, increases. Another issue common to the methods in [4]-[8] is that they do not elaborate on the disassembly of conditional branches; such branches requires careful consideration during both phases of disassembly and can enable the detection of possible transitions to different parts of the code and the evaluation of control flow for comprehensive disassembly. Finally, the methods in [4]-[7] extensively instantiate instructions with randomized operands, in different sequences, etc.; they

instantiate each instruction from 200 [6] to 3000 [5] times. These methods cannot be directly extended to fine-grained EM SCA because their acquisition costs would be infeasibly high, especially if the number of possible instructions and measurement configurations is large. By contrast, our proposed method aims to (i) improve the success rate of disassembly for application codes, (ii) identify if branches were taken/not taken during execution, and (iii) maintain a feasible acquisition cost even for large instruction sets and high-resolution EM probing.

2.2 Proposed Approach

As mentioned in the Introduction, the proposed method consists of two phases (Fig. 2). In the *feature-selection phase*, EM fields emanated from the DUT are collected for all instructions by designing and using profiling codes that instantiate each instruction for multiple specific machine states, chosen according to the HW leakage model [9], [15]. The signals are collected with all measurement configurations in a 5-D search space consisting of the probe location, probe orientation, and time interval. Next, the min-max bounds of signals—directly probed fields, as well as differential signals derived from them—are found for each instruction, and these signal envelopes are compiled within a hierarchical database. The database stores for each instruction—at the bottom stage of the hierarchy—real-valued envelopes that are multivariate functions of the measurement configuration, i.e., they are functions of 5 variables. For the upper stages of the hierarchy, instructions are grouped using certain instruction attributes (Fig. 1), and the database is compiled bottom-up, i.e., the envelopes for the instruction classes in the upper stages are constructed using envelopes for instruction classes compiled in the lower stages. Once the database is constructed, it is used to identify optimal measurement configurations and features for binary classification. During feature selection, the envelopes for each instruction class are compared pairwise (one at a time) to those of other classes at the same stage; the comparison identifies M configurations, where the pair's signal envelopes are most distant; i.e., these are the optimal values of the 5 variables to distinguish the pair from each other. The signals obtained with the optimal measurement configurations, i.e., the selected features, and the envelopes of the two classes corresponding to them are recorded for use in the next phase. In the *classification phase*, signals measured while the DUT executes arbitrary codes are categorized hierarchically starting from the top stage. At each stage, candidate classes are identified given the class selected in the previous stage, using binary classification with majority voting [5].

3 BACKGROUND

This section describes the DUT's measurement setup, the SCA threat model, the hierarchical grouping of the instruction set, and the signals used in the proposed method.

3.1 Measurement Setup

To demonstrate the proposed method, this article uses the AT89S51 microcontroller, which implements 111 instructions, differing in function, size, length, addressing mode, source and destination operands, etc. [12]. The setup used for the measurements is shown in Fig. 3. The DUT was operated at 2 MHz. Fields were sensed using a 1-mm H-field probe, positioned at a fixed height of 0.5 mm and various points on an equally spaced 51×51 grid over the DUT's surface (area~8×8 mm^2) using Riscure's probe positioner. Measurements were performed using both x- and y- oriented probes. Therefore, N_{pc} =5202 probe configurations were used for constructing the database. Signals were collected and analyzed using a Keysight DSOS054A oscilloscope, at a sampling rate of 2 GS/s ($N_{\mathrm{t}} = 1000$ samples); the signals were collected 50 times and averaged to minimize measurement noise. For comparison and validation, measurements using the coarse-grained EM SCA setup were also performed, using a 10-mm H-field probe. HEX files for programs, generated using Keil's 8051 emulator, were uploaded to the program memory of the chip using an Arduino as interface. These codes included

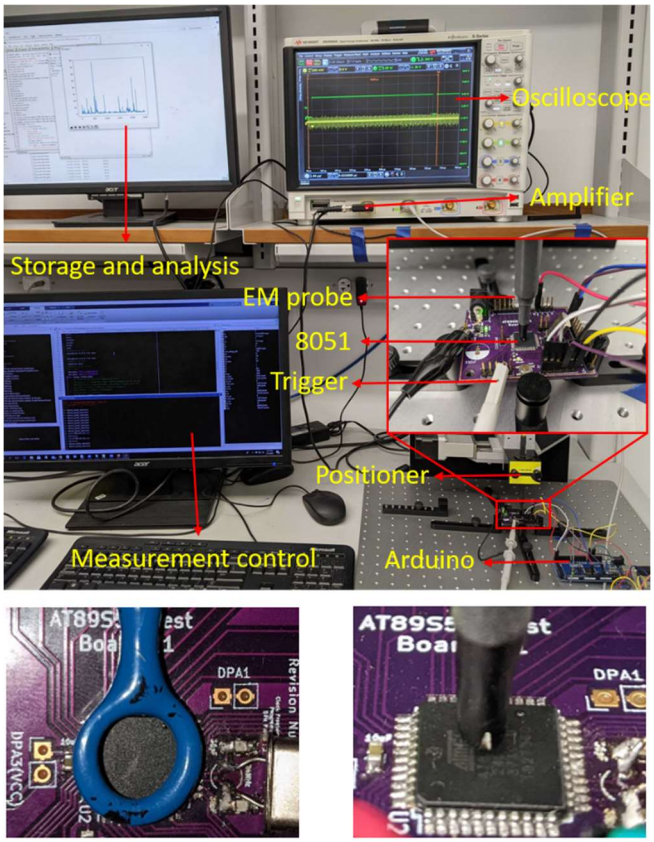


Fig. 3. Measurement setup used for instruction disassembly (top, same as in [9]) and probes used for coarse-grained (bottom-left) and fine-grained (bottom-right) EM SCA.

start/end markers to simplify measurements, implemented via a general-purpose I/O pin. The probe positioning, data acquisition, and subsequent data storage were automated to save experiment time. To reduce storage requirements, samples were saved as single-precision floating-point numbers in binary file format. More information on the setup can be found in [15], [16]. Only $N = 90$ instructions were considered for the following analyses; instructions that use external and indirect addressing modes were excluded because such instructions are seldom used by compilers for general-purpose codes, unless access to external memory is required, and because the focus of this article is on EM emanations arising from on-chip switching activity.

3.2 Threat Model

Different threat models are assumed in the feature-selection and classification phase experiments. To allow accurate profiling, limited restrictions are placed on evaluators during the first phase. As in previous works [4]-[8], the feature-selection phase assumes that evaluators have the ability to control a clone of the DUT, or the DUT itself such that they have the ability to send known profiling codes to the device and observe the internal architectural state of the microcontroller as each instruction is executed. Further, the evaluators are assumed to also have the ability to repeat such

Table 2: Instruction Classes

Length	Size	Operands	Functions
1Cycle (51 ins)	1Byte (25 ins)	Acc[1]	INC; DEC; RR; RRC; RL; RLC; SWAP; DA; CPL; CLR
		Acc,Reg	ADD; ADDC; SUBB; ORL; XRL; ANL; MOV; XCH
		C-bit[2]	SETB; CLR; CPL
		Reg[3]	INC; DEC
		Reg,Acc	MOV
		No ops.	NOP
	2Byte (26 ins)	Acc, Imm[4]	ADD; ADDC; SUBB; ORL; XRL; ANL; MOV
		Acc, Dir	ADD; ADDC; ORL; ANL; XRL; SUBB;MOV; XCH
		Dir[5]	INC; DEC
		C-bit, Bit	MOV
		Bit[6]	CLR; CPL; SETB
		Reg, Imm	MOV
		Dir, Acc	ORL; ANL; XRL; MOV
2Cycle (51 ins)	1Byte(5 ins)	Acc, Dptr[7]	MOVC
		Acc, PC[8]	JMP; MOVC
		No ops.	RET;RETI
	2Byte (17 ins)	Addr[9]	ACALL; AJMP
		C, Bit	ANL; ORL
		Reg, Off[10]	DJNZ
		Off	JZ; JNZ; JC; JNC; SJMP
		C, /Bit	ANL; ORL
		Dir	PUSH;POP
		Reg, Dir	MOV
		Dir, Reg	MOV
		Bit, Cbit	MOV
	3Byte (15 ins)	Dir, Imm	MOV; ANL; ORL; XRL
		Bit, Off	JB; JBC; JNB
		Addr	LCALL;LJMP
		Acc, Imm, Off	CJNE
		Acc, Dir, Off	CJNE
		Reg, Imm, Off	CJNE
		Dir, Off	DJNZ
		Dir, Dir	MOV
		Dptr, Imm	MOV
4Cycle (2 ins)	1Byte (2 ins)	Acc, B[11]	MUL;DIV

[1]Accumulator, [2] Carry Bit, [3] General Purpose Registers, [4] Immediate Value,[5] Direct RAM Address, [6] Register Bit, [7] Data Pointer, [8] Program Counter, [9] Branch Address, [10] Branch Offset, [11] B Register

codes as many times as desired, allowing field measurements to be averaged to minimize measurement noise. In contrast to this transparent “white-box” model of the feature-selection phase, a more restrictive “gray-box” model [17] is used in

the classification phase. In this model, the code being executed, the inputs, and the internal operations of the DUT are assumed to be not visible to the evaluators but the evaluators are assumed to still have the ability to repeat the codes being targeted, similar to the setup used by other fine-grained EM works that combine measurements from multiple locations to increase success rates of disassembling instructions [7], [8], or identify an instruction's functional units [18].

3.3 Hierarchical Grouping of Instructions

Attempting to directly classify measured signals within a large set of candidate instructions increases the odds of misclassification. Hierarchical classification can decrease the misclassification risk by reducing the number of possible candidates in each stage, assuming the stages in the hierarchy are appropriately chosen for the DUT (poor groupings can result in potentially more misclassifications at the upper stages). In [5], a 2-stage hierarchy was used: the instructions were separated into 8 groups based on operands and into sub-groups based on their function. That grouping is not suitable for microcontrollers that have a large number of possible operands (>30 for AT89S51). Instead, in this article, 2 higher stages, where instructions are grouped according to length and size, are added to the hierarchy. In Stages III and IV of the hierarchy, instructions are grouped based on operands and their functions as in [5], resulting in 4 stages of hierarchy (Fig. 1). These 4 attributes of each instruction ins are represented with the label $\mathrm{ID}_{ins} = (L, S, Op, Fn)$. Here, L denotes the length, S the size, Op the operands, and Fn the function of the instruction i.e., how long it requires to complete execution, the number of bytes fetched from program memory for it, the memory locations of the chosen data values in it, and the operations it performs, respectively. In AT89S51, instructions require $L \in \{1,2,4\}$ cycles for execution, are of size $S \in \{1,2,3\}$ bytes, have 30 possible operands, and implement 45 functions. Table 2 shows the resulting hierarchy. In the following, cycle lengths and sizes are represented with the suffixes C and B; e.g., the label for the 1 cycle 1 byte instruction INC Acc is $\mathrm{ID}_{INC\ Acc}$ = (1C, 1B, Acc, INC).

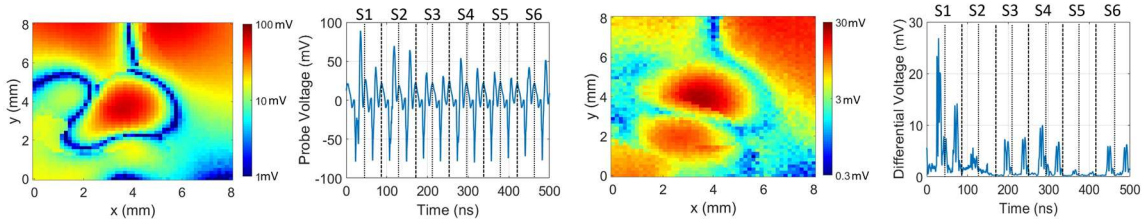
3.4 Observed Signals and Target Processes

Signals collected by a near-field probe above a DUT are functions of 5 variables in the measurement setup used (Fig. 3): The probe's configuration pc—its transverse location (x, y), height h, and orientation o relative to the DUT—and the time of observation t. Thus, the probed fields can be represented as 5-dimensional functions $V(pc, t)$. Of course, the measured signal also depends on the processes pr that the DUT is executing, i.e., the state of the microcontroller. These processes are performed at specific time-intervals within a DUT's machine cycle, localizing features temporally. The processes can be abstracted as a combination of a target process Tpr_i and one or more background processes Bpr_j, where the subscripts i and j represent versions within these processes [9]; e.g., if the entire instruction opcode is considered the target process, then the 90 target versions are Tpr_1 ≡INC Acc, Tpr_2 ≡ DEC Acc, …, Tpr_{90} ≡DIV Acc, B and the background processes include data operations in various architectural registers. The background processes can be represented using the state of architectural registers $X \in \{\mathrm{X}_1, \ldots\ \mathrm{X}_{N_\mathrm{x}}\}$, where each state X_k represents a unique data value in registers (RAM, stack, program counter, etc.) and N_x is the number of combinations of register contents. Thus, the signals can also be represented as 7-dimensional functions $V(pc, t, Tpr_i, Bpr_j)$. Using the notation in [9], a signal's dependence on measurement configuration and processes executed on the DUT are highlighted with super/sub-scripts; e.g., $V^{pc,t}_{Tpr_i,Bpr_j}$.

In addition to the probed fields $V^{pc,t}_{Tpr_i,Bpr_j}$, the differential signal

$$\Delta V^{pc,t}_{Tpr_i,Bpr_j} = \left| V^{pc,t+\Delta t}_{Tpr_i,Bpr_j} - V^{pc,t}_{Tpr_i,Bpr_j} \right|, \qquad (1)$$

is introduced. Here, Δt is the product of cycle length L of the target process Tpr_i and clock period T_{clk}. In this work, the differential signals are computed between the corresponding clock cycles of adjacent instructions. While traditional differential side-channel analysis assumes observed signals in a single clock cycle represents the transition between



(a) Probed Fields (b) Differential signals

Fig.4. Space-time distribution of (a) probed fields, and (b) differential signals derived from them, measured by a *y*-oriented probe at 51×51 locations for MOV A, #00 instruction. Spatial maps are plotted at 25 ns and time variations are plotted at the center location. Each machine cycle is divided into 6 states and 2 sub-states [12].

different machine states, the differential signal introduced in this article computes differences in fields over multiple clock cycles, i.e., it captures the change in fields measured from before an instruction is executed, to after it is executed. This is a useful quantity for separating instructions that modify contents of architectural blocks shared across the instruction set, such as program counters, or the pre-fetched architectural registers. For instance, the 8051 reserves certain sub-cycles to operate on the accumulator or certain RAM registers [11], irrespective of the executed instruction, enabling easier identification of instructions impacting these registers with differential signals. Example signals are plotted in Fig. 4.

If a single-stage disassembler was used, the target process would be the complete instruction opcode. Thus, each version of the target process from Tpr_1 to Tpr_{90} would represent a candidate opcode for disassembling the observed signals. The large set of candidates poses major issues in feature selection and classification; e.g., a total of ${}_{2}^{90}C$=4005 classifiers are required for binary classification [5]. In contrast, the proposed 4-stage hierarchical disassembler constructs only 281 classifiers because there are relatively small numbers of candidate classes in each stage. What constitutes target and background processes, however, changes at each stage of the hierarchy. The target process in each stage is a different attribute of the opcode, identified by the label $\mathrm{ID}_{ins} = (L, S, Op, Fn)$. Because classification in each stage distinguishes instructions based on only one attribute, the remaining attributes of the opcode are assumed to be part of the background: In Stage I, the target instruction length can take values from the set $L \in \{1\mathrm{C}, 2\mathrm{C}, 4\mathrm{C}\}$. Here Bpr for $L = 1$ C instructions includes any combination of the architectural state X, and the 51 groups of $(1\mathrm{C}, S, Op, Fn)$ in Table 2. The hierarchy then enables independent analysis within each branch in the following stages; e.g., in Stage II, the instruction size is analyzed separately for 1C instructions (for which $S \in \{1\mathrm{B}, 2\mathrm{B}\}$) and 2C ones (for which $S \in \{1\mathrm{B}, 2\mathrm{B}, 3\mathrm{B}\}$). Although attributes (S, Op, Fn) are assumed to be "background" processes here, they are still constrained by target process versions being evaluated, unlike the state of background architectural registers that is unrestricted.

4 PHASE I: FEATURE SELECTION

This section details the database construction, the profiling codes, and the feature-selection method in the first phase of disassembly.

4.1 Database Construction

Each instruction class is characterized by 4 signal envelopes in the database; these envelopes are 5-dimensional functions (of pc, t). The hierarchical database is constructed as follows (see Fig. 1 for stage definitions). First, the Stage IV portion

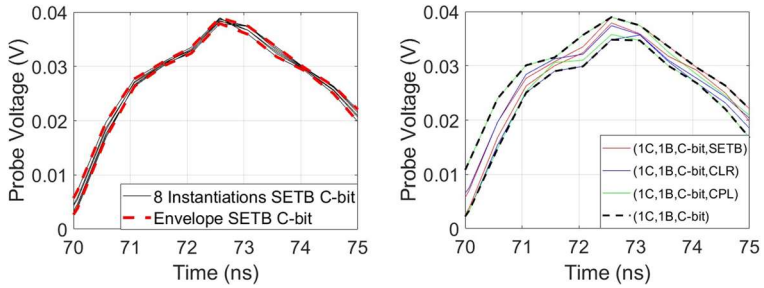


Fig.5. The envelopes in stage IV portion of the database (left) are the min-max bounds of the probed fields for multiple instantiations of each instruction; here, the SETB C-bit instruction. The instantiations have different initial conditions of the C-bit (0 and 1) and RAM registers (0x00 and 0xFF). The envelopes in stage III portion of the database (right) are the min-max bounds of the envelopes of all instructions that have the same operand; here, C-bit.

of the database is compiled for the 90 instructions. For each instruction Tpr_i, multiple instantiations are executed (see Section 4.2), the EM fields are probed using all possible probe configurations, and the min-max envelopes of probed fields and differential signals are stored in the database:

$$\mathbf{env}_{Tpr_i}^{pc,t} = [\min V\,, \max V\,, \min \Delta V\,, \max \Delta V] \qquad (2)$$

Here, the minima and maxima are found among all instantiations of the instruction, i.e., $\forall Bpr_j \in Bpr$. Next, these 90 instructions are grouped according to their operand class, as per Table 2. The envelopes for each of the 35 operand classes in Stage III are constructed by computing the min-max bounds of the envelopes of all the instructions with that operand. Similarly, Stage II (I) portions of the database are compiled from its Stage III (II) portions. Fig. 5 shows an example computation of the min-max envelopes.

4.2 Profiling Codes

One approach to finding the signal envelopes is to collect an extensive set of signals, e.g., by instantiating the architectural registers X with random values. For instance, [5] used 3000 such instantiations per instruction for feature selection. While this can improve classification accuracy for coarse-grained EM/power SCA setups, the acquisition cost for fine-grained EM setups quickly becomes intractable when so many instantiations are used: For $N = 90$ instructions, if $N_{\mathrm{t}} = 50$ time samples of signals are measured as in [5] with a single probe configuration ($N_{\mathrm{pc}} = 1$), a total of 13.5×10^6 samples would be acquired. If they are measured with the fine-grained EM SCA setup in this work, with N_{pc}~5200 probe configurations (Section 6.1), a total of 70×10^9 samples would be acquired. Storing these samples as single-precision floating-point numbers would require ~50 MB of space for the former and ~280 GB for the latter setup. Additional storage may be required during feature selection, e.g., to transform time-domain data to frequency domain.

A smaller set of signals can be collected by modeling the leakage as if it depends only on HWs of data in architectural registers, a common approach in processor security evaluations [9],[11]; e.g., signals for 256 data values can be bound by those for extreme instantiations of data 0x00 (HW 0) and 0xFF (HW 8). Then, the data-dependency of each instruction—except conditional branch instructions—can be bound by using at most 4 instantiations, by setting operands and result to data values 0x00 and 0xFF. For example, consider the instruction ADD Acc, Imm. To bound its data dependence, the data values in the Accumulator register and the Immediate value in program memory are chosen from the set {(0x00,0x00), (0x00,0xFF), (0xFF,0x00), (0xFF,0xFF)}. Further, to improve coverage of background processes, all 128 bytes of RAM, including stack registers, are instantiated as either 0x00 or 0xFF. Therefore, 8 instantiations are used to characterize each

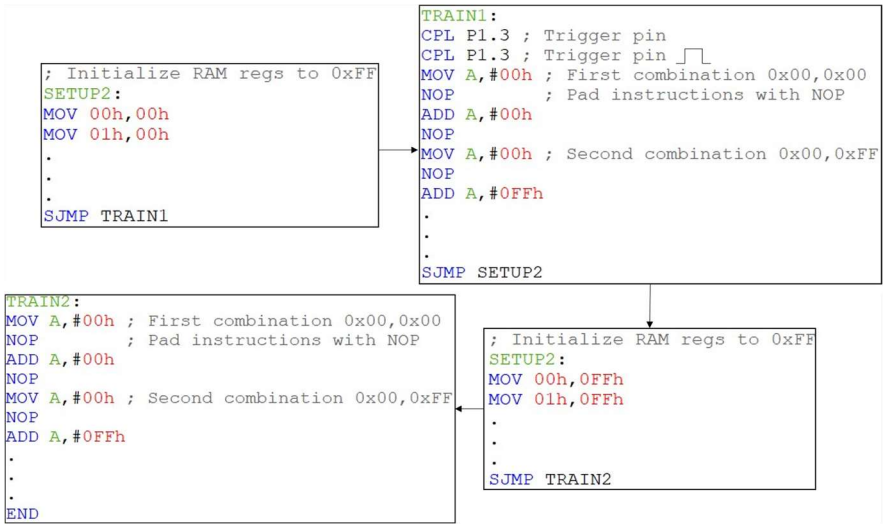


Fig.6. Profiling codes instantiate instructions with different operands, under different machine states. NOP instructions are introduced to keep the computation of differential signals consistent.

instruction in the profiling codes. Code snippets used to profile this instruction are shown in Fig. 6. In addition to the instruction instantiations, extra instructions are used to support measurements, such as a general-purpose pin triggering the oscilloscope for ease of experiment.

Because conditional branches perform different functions depending on the result of the condition evaluation, branches taken and not taken for the same instruction are considered as separate classes in Stage IV, i.e., they have the same instruction length, size, and operands, but different functions. Introducing 12 additional instruction classes for the conditional branch instructions in Table 2, control-flow prediction is enabled in the final stage of disassembly. Using 16 instantiations for conditional branch instructions and 8 for other instructions, the proposed profiling codes contain a total of $N\bar{N}_{\text{inst}} = 12 \times 16 + 78 \times 8 = 816$ specially-designed test instructions (in addition to miscellaneous instructions used as markers for measurement, and various instructions needed to clear flag registers, data memory, or stack). These profiling codes are used to acquire the following total number of samples to construct the database:

$$N_{\text{samp}} = N\bar{N}_{\text{inst}} N_{\text{pc}} N_{\text{t}} \quad (\#\text{ of Samples Acquired}) \qquad (3)$$

Here, N_{pc} is number of probe configurations, N_{t} is number of time samples, N is the number of instructions, and $\bar{N}_{\text{inst}}$ is the average number of instantiations used to profile each instruction. While $N_{\text{pc}}N_t$ depends on the measurement setup, $\bar{N}_{\text{inst}}$ depends on the profiling method.

4.3 Selecting the Features

Feature selection identifies optimal measurement configurations where envelopes (and therefore signals) are easily separable when compared pairwise. Here, as well as in Section 5, the process is presented for two instruction classes a and b at the same stage of the hierarchy. First, the "average distance" between the pairs' envelopes is computed:

$$Dist_{a,b}^{pc,t} = \frac{\left|(\mathbf{env}_a^{pc,t}[1]+\mathbf{env}_a^{pc,t}[2])-(\mathbf{env}_b^{pc,t}[1]+\mathbf{env}_b^{pc,t}[2])\right|}{2} \qquad (4)$$

While feature selection in Stages II-IV directly uses this quantity, a pre-processing step is required in Stage I because signals with different time lengths are compared. It is assumed that the first cycle of multi-cycle instructions is similar to

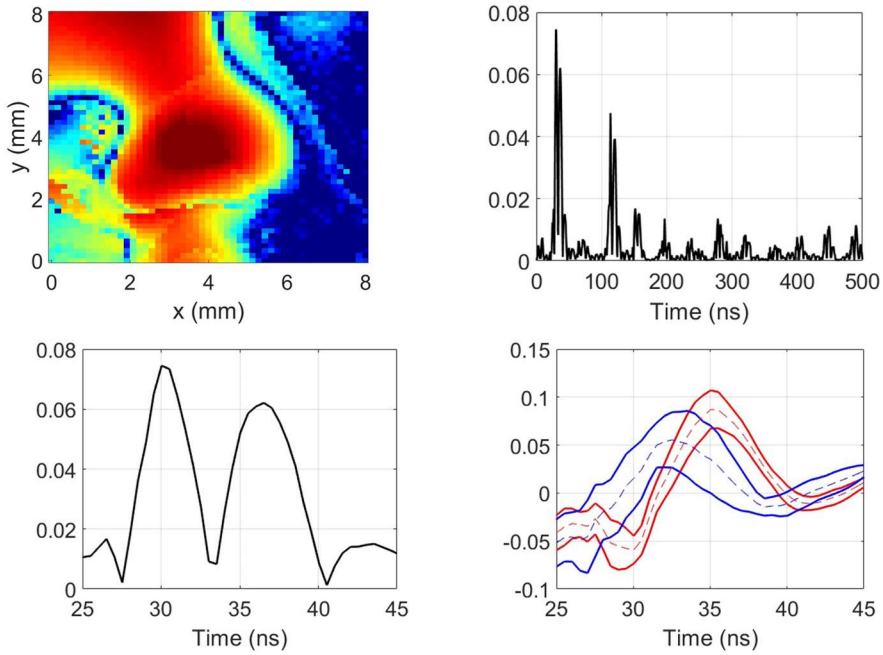


Fig.7. Spatial map (top-left) of $Dist_{1C,2C}^{pc,t}$ between 1-cycle and 2-cycle instructions at t~30 ns and time variation (top-right) at an optimal probe location (starred). Distance (bottom-left) and envelope (bottom-right) plots for an optimal time interval showed that instruction classes were more separable when the difference between the envelope averages (dashed) increased, particularly at t~30 and t~37 ns.

a single-cycle instruction, due to the presence of opcode fetch-related processes. Consequently, in Stage I feature selection, signals for multi-cycle instructions are partitioned into multiple single-cycle windows, similar to [4]. The partitioned windows are then compared separately to single-cycle instructions, assuming the cycles that follow the first cycle will show sufficient differences to allow their length-based classification. Fig. 7 shows an example of the distance between single-cycle instructions and the second cycle of two-cycle instructions. The distance $\Delta Dist_{a,b}^{pc,t}$ between the differential signal envelopes is computed similarly. As demonstrated in Fig. 8, some instruction classes are potentially more separable using differential signals. Prediction of a program's control flow can be achieved in Stage IV of the disassembly, as shown in Fig. 9.

Next, optimal measurement configurations that maximize the distance between signal envelopes are identified. For each pairwise comparison, $M = 10$ optimal probe configurations—5 each for direct and differential signals—and the corresponding 10 optimal time instances are stored in the arrays $\mathbf{pc}_{a,b}^{\text{opt}}$ and $\mathbf{t}_{a,b}^{\text{opt}}$. The signals at these optimal measurement configurations are the selected features that will be compared with the stored envelopes to classify instructions.

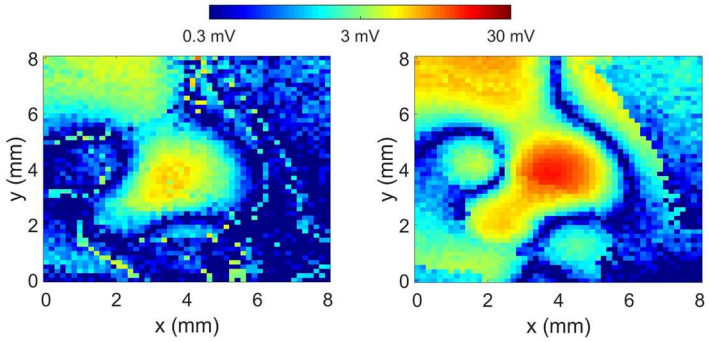


Fig.8. Comparing the classes $(1C, 2B, \mathrm{Dir})$ and $(1C, 2B, [\mathrm{Acc}, \mathrm{Dir}])$ in stage III with $Dist_{a,b}^{pc,t}$ (left) and $\Delta Dist_{a,b}^{pc,t}$ (right) shows that they are more separable when using differential signals. Here, t~120 ns.

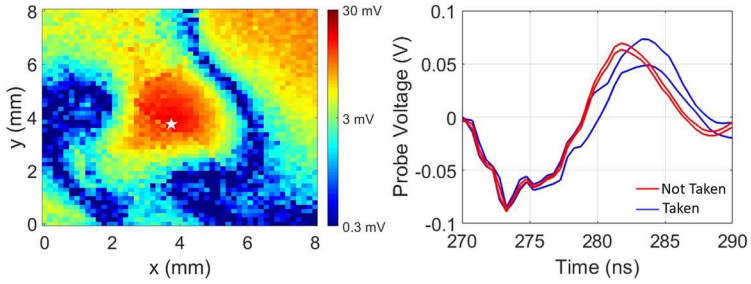


Fig.9. Distance between branch "taken" and "not taken" classes for instruction $(1C, 2B, \mathrm{Off}, \mathrm{JNZ})$ in Stage IV (left), shows that the disassembly can potentially predict program flow. The spatial map of distance is plotted at t~285 ns and the observed fields are plotted at an optimal configuration (starred).

5 PHASE II: CLASSIFICATION

During classification, the probed field $V^{pc,t}$ and differential signal $\Delta V^{pc,t}$ are compared to the signal envelopes in the database. The deviation of evaluated signals from the envelopes of candidate classes a and b in the database are computed as

$$Dev_{a/b}^{pc,t} = \mathrm{Max}\left\{V - \mathbf{env}_{a/b}^{pc,t}[2], 0\right\} + \mathrm{Max}\left\{\mathbf{env}_{a/b}^{pc,t}[1] - V, 0\right\} \qquad (5)$$

This metric is 0 if the evaluated signal is within the stored envelope. The deviation of a probed field from the envelopes in Fig. 7 is shown in Fig. 10. A corresponding metric $\Delta Dev_{a/b}^{pc,t}$ is computed for the differential signals.

During binary classification, the net deviation of the evaluated signal from the two candidates a and b is computed only with the M optimal measurement configurations for separating them:

$$NetDev_{a/b} = \sum_{m=1}^{M/2} Dev_{a/b}^{\mathbf{pc}_{a,b}^{\mathrm{opt}}[m], \mathbf{t}_{a/b}^{\mathrm{opt}}[m]} + \sum_{m=M/2+1}^{M} \Delta Dev_{a/b}^{\mathbf{pc}_{a,b}^{\mathrm{opt}}[m], \mathbf{t}_{a,b}^{\mathrm{opt}}[m]} \qquad (6)$$

The instruction class with the smaller net deviation is considered the more likely candidate for the evaluated signal. To classify among multiple candidates, the binary classification is implemented with a majority voting method [5]:

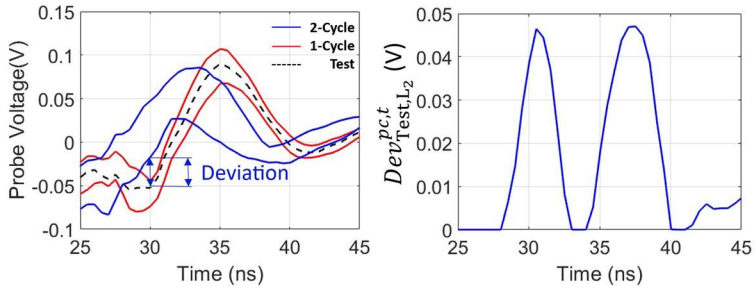


Fig.10. An evaluated signal for instruction (1C,1B,Acc,Inc) correctly shows large deviation from envelope of 2-cycle instructions at t~30 ns and t~37 ns.

$$vote_{a,b} = \begin{cases} +1, \text{if } NetDev_a \geq NetDev_b \\ -1, \text{if } NetDev_a < NetDev_b \end{cases}$$
$$a^* = \underset{a}{\operatorname{argmax}} \sum_{b=1\,(b\neq a)}^{N_c} vote_{a,b} \qquad (7)$$

Here, a^* is the most likely candidate class and N_c is the number of candidate classes.

6 EXPERIMENTS AND RESULTS

To test the proposed disassembler, first, each instruction is instantiated 100 times with random operand values. In this test set, each instruction is padded with a NOP instruction, and before the instantiations the RAM registers are cleared, similar to the profiling codes shown in Fig. 6. A total of 10200 instructions are evaluated in this test set. This evaluation is similar to the test sets that follow the templates of profiling codes, used in [4]-[7]. For conditional branch instructions, two separate test sets are used for the branch "taken" and "not-taken" cases. The operands in both cases are randomized with constraints, to ensure the functions are correctly executed; e.g., for the jump-if-not-zero instruction's branch "taken" case, the operand is allowed to take all values other than 0.

Second, a more robust and complete evaluation of the proposed disassembler is performed by using a set of 4 application codes from Dalton benchmarks [14], which are specifically designed to optimize the performance of 8051 cores: the greatest common divisor (GCD), Fibonacci (FIB), sort, and square root (SQRT) codes. As their names indicate, the codes compute the GCD of two numbers, generate the first 10 Fibonacci numbers, sort 10 specified integers in ascending order, and find the square root of a specified floating-point number. The compiled codes were first disassembled using KIEL's 8051 emulator, providing a reference assembly code to judge the accuracy of the proposed disassembler.
Third, the potency of fine-grained EM SCA approach is evaluated by implementing the proposed feature-selection and classification methodology using a coarse-grained EM SCA setup (with a relatively large probe [6]) and comparing the success rates of the two approaches. Here, the measurement configurations are optimized only over the time dimension as there is a single fixed probe location and orientation.

6.1 Feature-Selection Results

To construct the database with the proposed profiling codes, a total of $N_{\text{samp}} = N\bar{N}_{\text{inst}}N_{\text{pc}}N_{\text{t}} = 816 \times 5202 \times 1000 \sim 4.2 \times 10^9$ samples (after they were averaged 50 times by the oscilloscope) were acquired. For comparison, consider

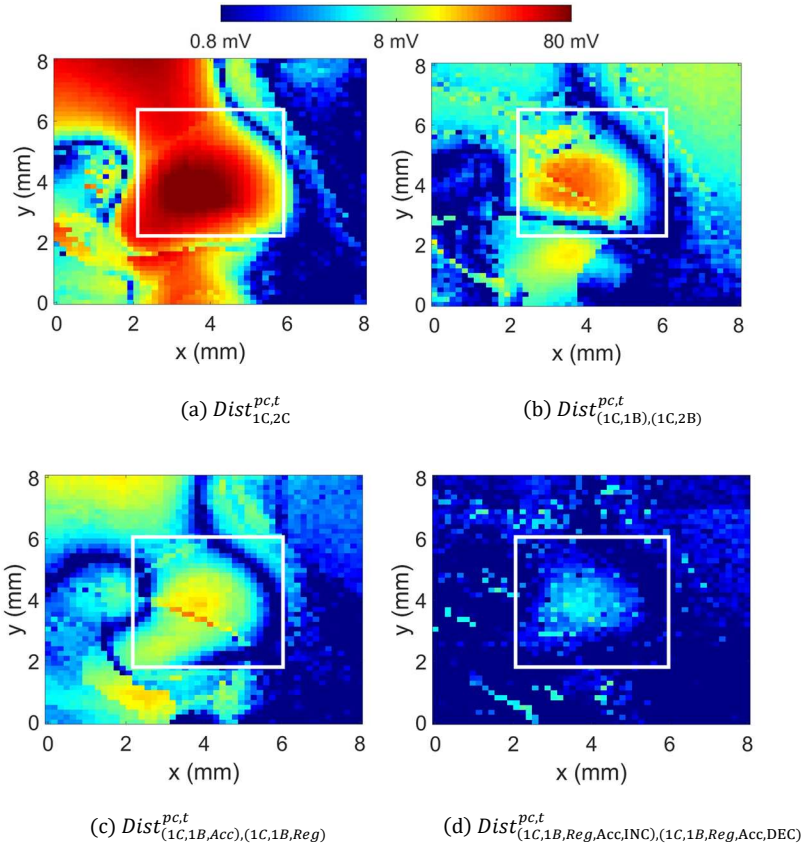


Fig.11. Example spatial maps of the envelope-to-envelope distances computed during feature selection phase in stages (a) I (t~30 ns), (b) II (t~270 ns), (c) III (t~360 ns), and (d) IV (t~70 ns), observed at the most optimal time instants. The distances between instruction classes are smaller at lower stages of the hierarchy.

applying the methods presented in [4]-[7] directly to the presented fine-grained EM SCA setup: Assuming N_{pc} and N are the same as in this work, but using the same $\bar{N}_{\mathrm{inst}}$ and N_{t} values as in the previous works, the methods would require ~222× [4], ~17× [5], ~2.2× [6], and ~650× [7] more samples than the proposed method.

Results for feature selection phase are exemplified in Fig. 11, which shows that the envelope-to-envelope distances reduce across space and time at the lower stages of the hierarchy. This behavior is expected for well-designed hierarchies that progressively refine the granularity of recovered instruction. It was also observed that the spatio-temporal distributions of distances for each stage were different, i.e., each stage of the hierarchy impacted the probed fields differently. Further, it was observed that features for all classifiers were limited to the region marked with white in Fig. 11. Consequently, measurements for the classification phase were limited to this region (25×25 locations).

6.2 Classification Results

First, the test codes with 100 randomized instantiations of each instruction were disassembled and the recovered results were compared to the reference assembly code line by line. The accuracy is then simply computed as a ratio of correctly

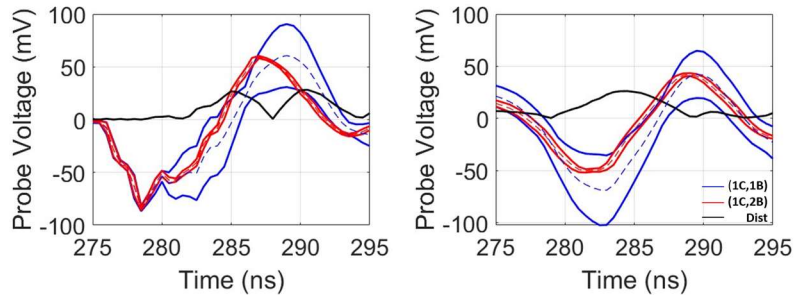
Table 3: Measurement Results

Benchmark	Code Size (bytes)	# of Instructions	Fine-Grained EM		Coarse-Grained EM	
			# of Correct Instructions	Accuracy (%)	# of Correct Instructions	Accuracy (%)
GCD	55	111	108	~97.3	71	~64.0
FIB	303	804	794	~98.7	531	~66.0
sort	572	2665	2556	~95.9	1702	~63.9
SQRT	1167	2006	1972	~98.3	1327	~66.1
Total	2097	5586	5430	~97.2	3631	~65.0

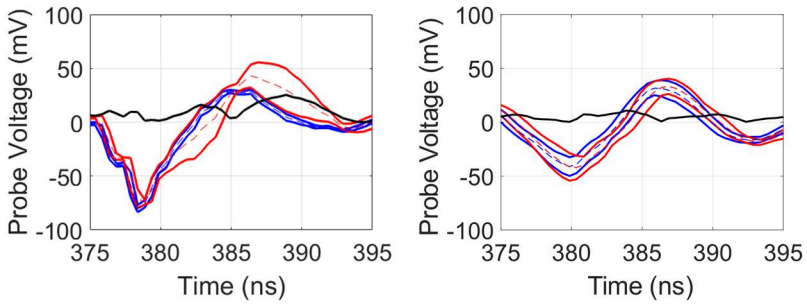
recovered instructions to the total number of instructions. The success rate of the disassembly was 10130 out of 10200 instructions (~99.3%). Evaluating accuracy stage-wise showed that the disassembled instructions had 100% accuracy for all instructions in Stages I-III, i.e., all misclassifications were in Stage IV. Therefore, the incorrectly recovered instructions still contained some relevant information. It was also observed that all conditional branches were correctly identified, including if the branch was taken or not. Such high success rates are to be expected because these codes follow a similar template to the profiling codes.

Results for the disassembly of application benchmarks are shown in Table 3. The total accuracy for the fine-grained setup was found to be ~97%, with less than $\pm 2\%$ variation among the 4 benchmarks. Similar to the evaluation of the test codes, no misclassifications were observed in the first three stages, and a 100% accuracy was observed in identifying conditional branch instructions. While a slight decrease in the disassembly accuracy was observed for the benchmarks, the difference is minimal compared to the disassemblers demonstrated in [4] and [7]. Finally, the most misidentified instruction for both test codes and benchmarks was the ADDC Acc, Reg, commonly misclassified as instruction ADD Acc, Reg (misclassified in 22 out of 123 instances). Potential reasons for the misclassification have to do with the close functional relation between the ADD and ADDC (i.e., add with carry) instructions, since in the absence of a carry bit, identical operations are performed by the microarchitecture. The opcodes of these instructions in the ISA are also very similar, including how they are decoded. Similar misclassifications were also observed for rotate and rotate with carry instructions that only differ minimally in functionality and operation. However, these instructions are not frequently used by the compiler thereby limiting inaccuracies and misclassification rates in large benchmarks.

The disassembler implemented using the coarse-grained EM SCA only showed a success rate of ~70% disassembling test codes and ~65% accuracy disassembling the benchmarks (Table 3). Contrary to the fine-grained measurement setup, misclassifications were observed in Stages II, III, and IV. Clearly, the fine-grained EM SCA setup resulted in a more potent disassembler. An example demonstrating the differences between database envelopes for the fine-grained and coarse-grained EM setups are shown in Fig. 12. It was observed that envelopes from the fine-grained setup were narrower and had sharper signal variations compared to the envelopes from the coarse-grained setup. Consequently the min-max envelopes predicted by the coarse-grained setup overlap for multiple classes at selected configurations leading to misclassifications, even when distance predicted between instruction classes is high (Fig. 12). Further, the overlap is also observed to increase in the coarse-grained case, as the classification moves to the lower hierarchical levels.



(a) Stage II comparison



(b) Stage III comparison

Fig.12. Signal envelopes and distance metric for fine-grained EM SCA setup at an optimal configuration (left) and coarse-grained EM SCA setup (right) for (a) instruction size-based classification in Stage II, separating single-cycle one-byte instructions and single-cycle two-byte instructions, and (b) operand-based classification in Stage III, separating the accumulator and register operands for single-cycle-one-byte instructions. Distance metric for fine-grained setup shows relatively sharp peaks, where the class envelopes can be clearly separated. By contrast, distance metric for the coarse-grained setup has broader peaks, with significant overlap of envelopes.

7 CONCLUSIONS AND FUTURE WORK

A fine-grained EM SCA based disassembler was proposed to recover instructions executed on a general-purpose micro-controller. The proposed method uses a hierarchical framework to improve feature selection and classification. It identifies optimal measurement configurations that distinguish instruction classes in the first phase by (i) executing model-based profiling codes to efficiently collect probed fields in a database, (ii) finding envelopes that bound the probed fields and, a novel quantity, differential signals derived from them. In the second phase, measured signals with these optimal measurement configurations are classified by comparing them to the signal envelopes of instruction classes one pair at a time. The comparisons were performed by quantifying the deviation of the measured signals from the signal envelopes. The proposed disassembler was shown to successfully and feasibly recover ~97% to ~99% instructions from application benchmarks and test codes executed on an AT89S51 microcontroller. Further, all conditional branch executions were correctly identified, enabling control-flow leakage prediction. It was also observed that the fine-grained EM SCA was significantly more potent compared to a coarse-grained EM SCA analysis.

The proposed disassembler can potentially detect malware within basic blocks [19], as well as those impacting control flow integrity [20]-[22]. Combined with appropriate tools quantifying vulnerabilities in side channels [15], [23]-[25], the disassembler can further enable programmers to optimize programs to minimize leakage. Finally, the instruction level

granularity of the disassembler enables detection of small-scale hardware trojans that are more challenging to address compared to malicious code [26].

The DUT used in this article simplifies the disassembly significantly because of its low-complex multi-cycle architecture; additional work is required to extend the proposed work to more complex embedded processors. For instance, in [27], randomized instructions were introduced based on the number of pipeline stages, while profiling individual instruction classes. A similar extension can be proposed for the fine-grained disassembler in this work; e.g., the feature-selection phase in heavily-pipelined processors can be split into two sub-phases: The first sub-phase can implement the feature-selection methodology, using a few select instructions padded with NOPs (Section 4.2). Once a sufficiently small set of potent probe configurations are identified, the NOP instructions can be replaced with randomized instructions and operands for reduction, depending on the number of pipeline stages. Additional datasets can also be created for groups with a large number of instructions, to improve their disassembly, similar to [27].

The disassembly can be improved further by recovering data values of operands [9], in addition to instructions. There is also potential to improve disassembly with higher-resolution probes. A more optimal method of combining features from multiple configurations can also reduce misclassifications, with the potential to re-examine predicted results and observe anomalies. Further, differential signals are a novel quantity that requires further exploration, potentially being used to observe changes across multiple pipeline stages as the instruction is executed, adding a new dimension to the analysis. Finally, imposing more restrictions on evaluators in the classification phase, similar to generic black-box testing threat models, may necessitate the use of more potent post-processing techniques in combination with some of the aforementioned potential improvements to the setup. Code monitoring through instruction disassembly presents a non-invasive pathway to detect intrusions, and therefore evaluate embedded hardware security.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation Grant CCF-1901446.

REFERENCES

[1] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On Code Execution Tracking via Power Side-Channel. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 1019–1031. https://doi.org/10.1145/2976749.2978299

[2] Robert Callan, Farnaz Behrang, Alenka Zajic, Milos Prvulovic, and Alessandro Orso. 2016. Zero-overhead profiling via EM emanations. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). Association for Computing Machinery, New York, NY, USA, 401–412. https://doi.org/10.1145/2931037.2931065

[3] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes 2014. Precise instruction-level side channel profiling of embedded processors. In Proceedings of the Information Security Practice and Experience (ISPEC 2014) Lecture Notes in Computer Science, vol 8434. Springer, Cham. https://doi.org/10.1007/978-3-319-06320-1_11

[4] Thomas Eisenbarth, Christof Paar, and Björn Weghenkel 2010. Building a side channel based disassembler. In Transactions on Computational Science X. Lecture Notes in Computer Science, vol 6340. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17499-5_4

[5] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte, and Mark Tehranipoor. 2018. Power-based side-channel instruction-level disassembler. In Proceedings of the 55th Annual Design Automation Conference (DAC '18). Association for Computing Machinery, New York, NY, USA, Article 119, 1–6. https://doi.org/10.1145/3195970.3196094

[6] Varghese M. Vaidyan and Akhilesh Tyagi. 2020. Instruction Level Disassembly through Electromagnetic Side-Chanel: Machine Learning Classification Approach with Reduced Combinatorial Complexity. In Proceedings of the 2020 3rd International Conference on Signal Processing and Machine Learning (SPML 2020). Association for Computing Machinery, New York, NY, USA, 124–130. https://doi.org/10.1145/3432291.3432300

[7] Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar. 2015. Scandalee: a side-channel-based disassembler using local electromagnetic emanations. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15). EDA Consortium, San Jose, CA, USA, 139–144.

[8] Valence Cristiani, , Maxime Lecomte,, Thomas Hiscock .2019. A Bit-Level Approach to Side Channel Based Disassembling. In proceedings of Smart Card Research and Advanced Applications (CARDIS 2019) Lecture Notes in Computer Science 11833. Springer, Cham. https://doi.org/10.1007/978-3-030-42068-0_9.

[9] Vishnuvardhan V. Iyer and Ali E. Yılmaz. 2021. Using the ANOVA F-statistic to isolate information-revealing near-field measurement configurations for embedded systems. In proceedings of the 2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium, 1024-1029. https://doi.org/10.1109/EMC/SI/PI/EMCEurope52599.2021.9559360.

[10] Vishnuvardhan V. Iyer and Ali E. Yilmaz. 2022. An ANOVA method to rapidly assess information leakage near cryptographic modules. IEEE Transactions on Electromagnetic Compatibility, 64, 4 (August 2022), 915-929. https:// doi.org/10.1109/TEMC.2022.3157664.

[11] Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. 2007. Power and electromagnetic analysis: improved model, consequences and comparisons. Integr. VLSI J. 40, 1 (January 2007), 52–60. https://doi.org/10.1016/j.vlsi.2005.12.013

[12] John Wharton.1980. An Introduction to the Intel MCS-51 SingleChip Microcomputer Family. Application Note AP-69 (May 1980), Intel Corporation

[13] ATMEL. 2008. 8-bit microcontroller with 4K bytes in-system programmable flash. AT89S51 datasheet..

[14] Dalton Project/ Benchmark applications for synthesizeable VHDL model. i8051 benchmarks. http://www.ann.ece.ufl.edu/i8051/i8051benchmarks/index.html

[15] Aditya Thimmaiah, Vishnuvardhan V. Iyer, Andreas Gerstlauer, Michael Orshansky. 2022. High-level simulation of embedded software vulnerabilities to EM side-channel attacks. In Proceedings of Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2022). Lecture Notes in Computer Science, vol 13511. Springer, Cham. https://doi.org/10.1007/978-3-031-15074-6_10.

[16] Vishnuvardhan V. Iyer. 2019. An adaptive measurement protocol for fine-grained electromagnetic side-channel analysis of cryptographic modules. M.S. thesis (August 2019), Univ. of Texas, Austin.

[17] Vishnuvardhan V. Iyer, Meizhi Wang, Jaydeep Kulkarni and Ali E. Yilmaz. 2021. A systematic evaluation of EM and power side-channel analysis attacks on AES implementations. In proceedings of the 2021 IEEE International Conference on Intelligence and Security Informatics (ISI 2021), San Antonio, TX, USA, 1-6.

[18] Julien Maillard, Thomas Hiscock, Maxime Lecomte and Christophe Clavier. 2022. Towards fine-grained side-channel instruction disassembly on a system-on-chip. In proceedings of the 25th Euromicro Conference on Digital System Design (DSD 2022), Maspalomas, Spain, 472-479.

[19] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. 2017. EDDIE: EM-Based Detection of Deviations in Program Execution. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17). Association for Computing Machinery, New York, NY, USA, 333–346. https://doi.org/10.1145/3079856.3080223

[20] N. Carlini and D. Wagner, “ROP is still dangerous: breaking modern defenses,” in Proc. USENIX, Aug. 2014.

[21] Nicholas Carlini and David Wagner. 2014. ROP is still dangerous: breaking modern defenses. In Proceedings of the 23rd USENIX conference on Security Symposium (SEC'14). USENIX Association, USA, 385–399.

[22] Tyler Bletsch, Xuxian Jiang, Vince W. Freeh, and Zhenkai Liang. 2011. Jump-oriented programming: a new class of code-reuse attack. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS '11). Association for Computing Machinery, New York, NY, USA, 30–40. https://doi.org/10.1145/1966913.1966919

[23] Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R. Gross. 2015. Control-flow bending: on the effectiveness of control-flow integrity. In Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15). USENIX Association, USA, 161–176.

[24] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: a metric for measuring information leakage. In Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12). IEEE Computer Society, USA, 106–117

[25] Robert Callan, Alenka Zajić, and Milos Prvulovic. 2014. A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events. In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47). IEEE Computer Society, USA, 242–254. https://doi.org/10.1109/MICRO.2014.39

[26] Hongbo Gao, Qingbao Li, Yu Zhu, Yong Liu. 2012. Code-controlled hardware trojan horse, In Proceedings of Information Computing and Applications (ICICA 2012) Communications in Computer and Information Science, vol 308. Springer, doi: 10.1007/978-3-642-34041-3_26.

[27] Van Geest, J. and Buhan, I., 2022. A side-channel-based disassembler for the ARM-Cortex M0. In Applied Cryptography and Network Security Workshops: ACNS 2022 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Rome, Italy, June 20–23, 2022, Proceedings (pp. 183-199). Cham: Springer International Publishing.

Received May 2023, Revised August 2023, Accepted October 2023.