FISCAL : Firmware Identification using Side-Channel Power Analysis

Deepak Krishnankutty, Ryan Robucci, Nilanjan Banerjee, and Chintan Patel
Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County

***Abstract*—Side-channel analysis techniques have been demonstrated for secret information extraction, vulnerability assessment and intrusion analysis. However, these techniques have not been applied to identify instructions running on a general purpose pipelined computing platform. In this work we identify instruction execution sequences on these platforms using side-channel power measurements. The technique uses a Principal Component Analysis (PCA) based model to generate multiple power-supply templates and a novel post-processing dynamic programming algorithm for optimal template matching. One unique aspect of this technique is that we take measurements on multiple power supply pins on the device, to increase the precision and accuracy. We apply our dynamic programming algorithm, to detect the sequence of execution clock cycles, on templates for single instructions that provides accuracy in the range of 69.2% to 87.5% for ten thousand observations, from individual power supply pins. We further augment this technique to generate multiple dictionaries for various length instructions and use data from multiple power supply pins to increases the accuracy in the range of 87% to 100%. Further, classification rates for instruction templates based on operand addressing mode and specific hardware used, range from 87% to 100% using as few as 10 principal components. Using our methodology, we can determine malicious insertions in a pre-determined code sequence with a 100% accuracy, as demonstrated in the results.**

I. Introduction

There has been a great emphasis on side-channel-based analysis to perform secret information extraction, vulnerability assessment and intrusion analysis. These include power-monitoring attacks [1] [2], timing attacks [3], electromagnetic attacks [4] [5], differential-fault-analysis attacks [6], scan-based attacks [7], cache-based attacks [8] [9], bus-snooping attacks [10] [11], and acoustic attacks [12] [13]. Existing techniques for side-channel analysis [14] [15] [12] do not provide fine-grained determination of code execution. Specifically, it is difficult to identify a sequence of *multiple-clock-cycle instructions* being executed on a general purpose pipelined computing platform.

Eisenbarth et. al. [14] proposed an instruction recovery strategy based on Hidden Markov Models (HMM) with a prediction accuracy between 35% and 70% for single instructions. Power Analysis measurements were performed on a PIC16F687 micro-controller at 1 MHz. In [16] and [17], the authors present an instruction classification technique with a 100% classification rate. However, this result was based on classification when the same instruction was compared against the templates generated for that particular instruction. In their work, the authors decomposed multi-clock-cycle instructions into several one-clock-cycle templates for template classification. They did not incorporate the entire range of instructions available on the micro-controller in their study, neither did they account for the multi-clock-cycle complex instruction set processing units.

This work was supported by the U.S. Office of Naval Research under Award N00014-15-1-2179

In this paper, we present a technique that generates power-supply templates using principle component analysis for individual instructions followed by a novel dynamic- programming algorithm that uses multiple template dictionaries to determine the *clock-cycle order* (sequence of execute-clock-cycles for a stream of instructions). We approach template classification with a strategy that operates on multi-clock-cycle instruction templates based on their *complex-instruction-set traits* and *number of execute clock cycles*. Further, we apply the technique to well-established code modules (for e.g. 128-bit AES encryption and PID controller) and present our prediction accuracy. Unlike [18], the technique presented in this paper is *independent of any specific sequence of execution*, thus providing a mechanism to detect malicious code insertions.

II. Experimental Setup

A custom board (Fig. 1) was designed and fabricated to measure power-consumption data during software execution. The board comprises of one control FPGA facilitating experimentation on a second FPGA, a Xilinx Spartan 3E. The Spartan 3E is the Device Under Test (DUT) and the design macro, an openMSP430 [19] (which emulates the general purpose pipelined computing platform) is instantiated on this FPGA. The openMSP430 instantiated on the DUT runs at a clock-rate of 10 MHz and implements various code sequences. The control FPGA functions as the control/communication device to convey debug and control data to the DUT. Four power-supply pins available on a TQFP package of the DUT are used for power supply side-channel analysis. Power-supply current probing is performed using 1 Ω-resistors in the supply path for each power pin. The voltage across each resistor due to the current consumption of the device is actively probed and amplified on-board and passed through coax cables to four channels on a Tektronix *DPO7354C* oscilloscope.

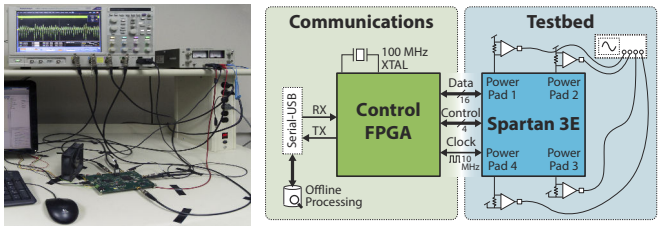


Fig. 1. Custom-designed board using FPGAs for communication/control and testbed. For the data presented in this paper, we used a high bandwidth 4-channel Tektronix *DPO7354C* oscilloscope.

III. Analysis Methodology

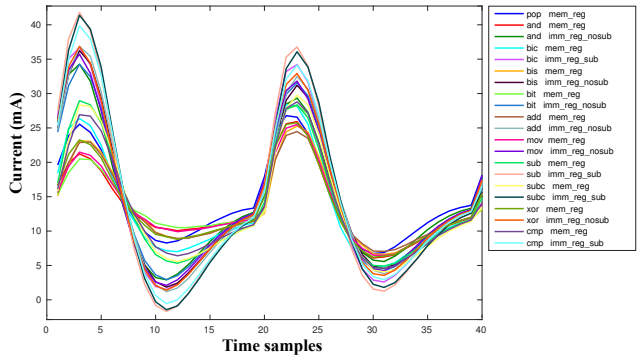
The openMSP430 instruction set consists of instructions that use 1- to 6-*clock-cycles per instruction* (henceforth referred to as N-CCPI where N is in the range 1 to 6) in the execute (EXE) stage as documented in [20].

A. N-CCPI Template Generation

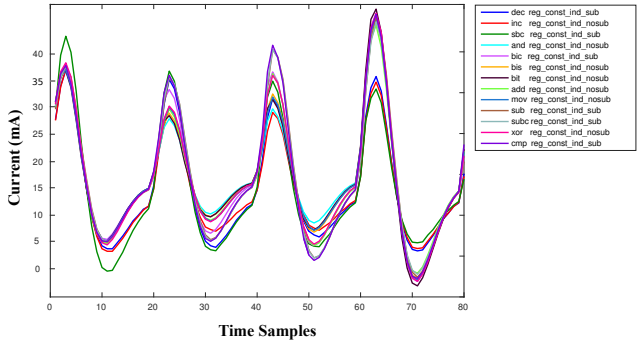
Power analysis of code execution on the openMSP430 instance was performed using data that sampled current signals

978-1-5090-4482-5/17/$31.00 ©2017 IEEE

at 200 MSPS which results in 20 time samples per clock-cycle on the oscilloscope. To generate the power-supply template for an instruction, a non-terminating loop sequence consisting of three instances of the instruction is observed for its power supply signature. Three instances account for non-linear variability of the observed instruction's power consumption over one iteration of the loop sequence. Each instance of an instruction being observed is succeeded by an operand update instruction that ensures the variability of the operands between different instances of the instruction being observed.



(a) Power profile for 2-CCPI templates



(b) Power profile for 4-CCPI templates

Fig. 2. Cases a) and b) depict the raw power profile averages for various clusters of 2-CCPI and 4-CCPI. Instructions were manually clustered based on hardware utilization.

For instructions that operate on memory, the register (initialized prior to the non-terminating loop) used for indexing is updated in the proceeding instruction. All non-emulated instructions in the MSP430 instruction set were utilized for constructing the training set. The averaged power profiles for 2-CCPI and 4-CCPI instructions, based on their execute-clock-cycles, over 20 000 instances derived from power-supply pin 1 are depicted in Fig. 2.

B. Principal Component Analysis of N-CCPI Templates

We utilize principal component analysis (PCA)[21] on the power-supply transient data, for dimensionality reduction and to avail the noise removal property when discarding lower-ranked components. We note that before PCA, each captured transient is prenormalized such that its RMS value is one and its mean is zero. To have an optimal template for comparisons between instructions, only the initial few (≈10) components are retained. The variance (squared eigenvalues) of principle components (Fig. 3) indicates that 10 PCA components are statistically sufficient to represent a power supply transient.

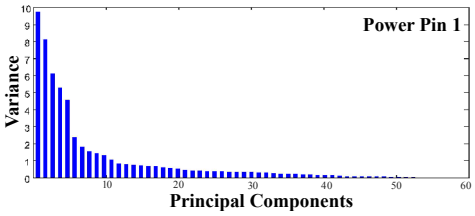


Fig. 3. The variance of principle components indicates that a power transient measured at supply pin 1 of a 3-CCPI instruction is statistically well-represented by the initial 10 PCA components

PCA is performed using all data from a given CCPI class. We use 20 000 raw data captures per instruction. This means that the number of transients used for PCA component determination for a given CCPI is $i \times 20\,000$ where i is the number of instructions with that given CCPI. Assuming the orthonormal principal components are stored in the columns of a matrix $\mathbf{P}$, the PCA coefficients $\mathbf{p}$ for a temporal waveform $\mathbf{x}$ are computed using $\mathbf{p} = \mathbf{P}^T x$. We define a distance metric for two waveforms x and y using the l_2-norm and PCA:

$$\mathrm{DPCA}(\mathbf{P}, \mathbf{x}, \mathbf{y}) = \left\| \mathbf{P^T x} - \mathbf{P^T y} \right\|_2 \qquad (1)$$

C. Hardware-Based Labeling for Classification

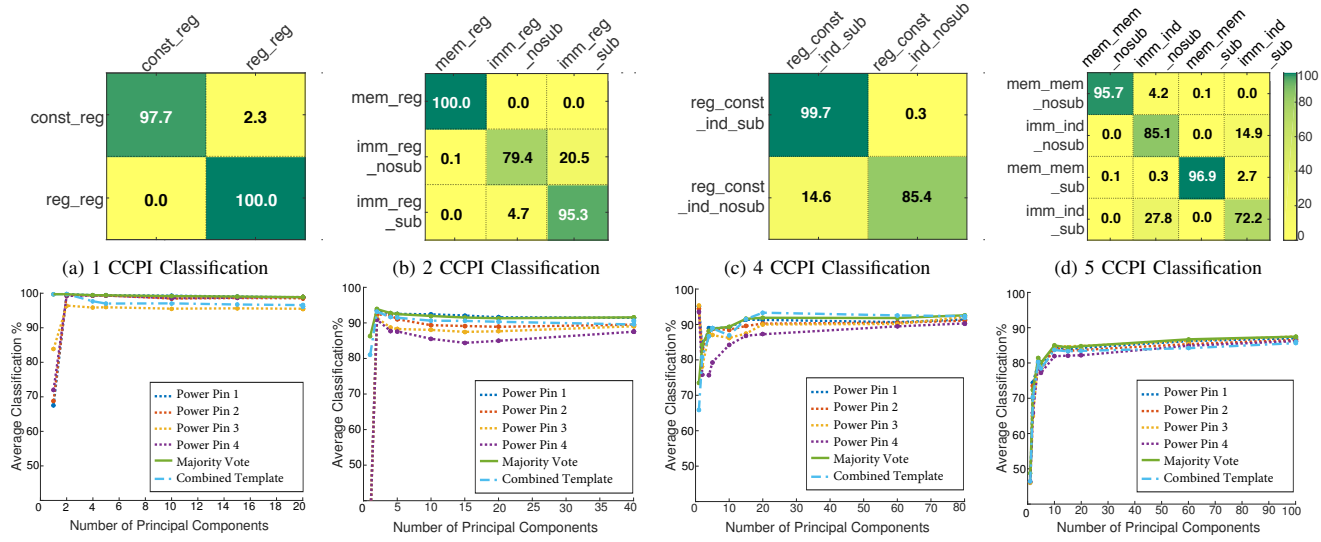
We manually group instructions based on hardware utilization and CCPI and choose to define a class label for each group. Hardware utilization (such as which hardware units are used) is affected by addressing mode (e.g. memory, register), computational operation, and status register updates. The class labels we choose, reflecting hardware utilization, are provided in Table I.

TABLE I
CLASSIFICATION OF INSTRUCTIONS BY THEIR HARDWARE USAGE TRAITS

#	Label	Interpretation (Associated Traits)
1	reg_reg	Source Register, Destination Register Includes Arithmetic and logical instructions
2	mem_mem_sub	Source Memory, Destination Memory (Complement hardware involved - includes 'bic' instruction)
3	mem_mem_nosub	Source Memory, Destination Memory (No complement hardware)
4	reg_const_ind_sub	Source Register or a constant, Destination Memory Destination includes a constant and involves indirect addressing
5	reg_const_ind_nosub	Source Register or a constant, Destination Memory Indirect addressing at source, No subtraction hardware
6	ind_reg_sub	Source Memory, Destination Register Indirect addressing at source, Subtraction hardware Used
7	ind_reg_nosub	Source Memory, Destination Register Indirect addressing at source, No subtraction hardware
8	mem_reg	Source Memory, Destination Register Includes Arithmetic and logical instructions
9	const_reg	Source is a generated constant, Destination Register Includes Arithmetic and logical instructions
10	imm_reg_sub	Source involves a constant, Destination Register (Complement hardware involved - includes 'bic' instruction)
11	imm_reg_nosub	Source involves a constant, Destination Register (No complement hardware)
12	imm_ind_sub	Source involves a constant, Destination Memory Indirect addressing involving constant at destination
13	imm_ind_nosub	Source involves a constant, Destination Memory Indirect addressing involving constant at destination, No complement hardware
14	other	Uncategorized

D. 1-NN Classification of a Single N-CCPI Instruction

In this section, a one-instruction classification using nearest neighbor (1-NN) classification is presented, while in the next section a more sophisticated algorithm for classifying heterogeneous instruction sequences is presented. For the 1-NN classification, training samples of the power supply transients for every instruction must be collected. 20 000 training samples of each instruction are captured by measuring the power-supply transient at every power pin on the FPGA package. Depending on the instruction's label we determined in Table I, we label



(a) 1 CCPI Classification (b) 2 CCPI Classification (c) 4 CCPI Classification (d) 5 CCPI Classification

(e) 1 CCPI Average Classification % (f) 2 CCPI Average Classification % (g) 4 CCPI Average Classification % (h) 5 CCPI Average Classification %

Fig. 4. Cases (a) to (d) - Confusion Matrices for N-CCPI instructions after 1-NN classification using labels in Table I on 20 000 test samples. At higher CCPI (cases (c) and (d)), the issue with the "bic" instruction reduces the classification rates for the "reg_const_ind_nosub", "imm_ind_nosub" and "imm_ind_sub classes". In cases (f), (g) and (h), higher instruction classification rates resulted when a majority decision was made on all four power supply pins.

and insert the training sample into the appropriate class record. For classification, we simultaneously perform a capture of the supply transient at each power pin, and then compare the test capture to all of the training. We select the class label according to the nearest training sample, using the distance metric, DPCA, defined in (1). The test sets used included 20 000 test samples per N-CCPI (1-, 2-, 4- and 5-clock-cycle) instructions. The results are summarized in Fig. 4(a-d). In Fig. 4(b), the classification rate for "imm_reg_nosub" class is 79.4%, while it is misclassified as "imm_reg_sub" 20.5% of the time. This is primarily attributed to the "bic" instruction, which complements the source operand, followed by a *bitwise AND* with the destination, thus being partly classified as an instruction that performs a 2's complement based subtraction. Misclassification rates are higher when the source operand is an immediate value. For 3- and 6-clock-cycle instructions, the average classification rates are 98.5% and 99.19% respectively. For Fig. 4(e-h), by retaining the first 10 PCA components, the average classification rates for N-CCPI templates tend to peak at 100% for 1-CCPI templates. Similar observations were made for 3-CCPI and 6-CCPI templates, and the classification rates peak at 98.7% and 99.7% respectively. A majority decision across the four power supply pins improves classification rates. Furthermore, to compare inter-pin relationships with prior observations, we construct a composite template by combining the templates from each power pin. A *10-fold cross-validation* was performed over a data set size of 50 00 instances for every instruction template, which confirmed the classification results (not included due to space constraints).

E. Classifying a Sequence of Heterogeneous-CCPI Instructions

The N-CCPI templates constructed from an optimal number of PCA components are grouped into a structure T which is utilized by the dynamic programming technique described in this section.

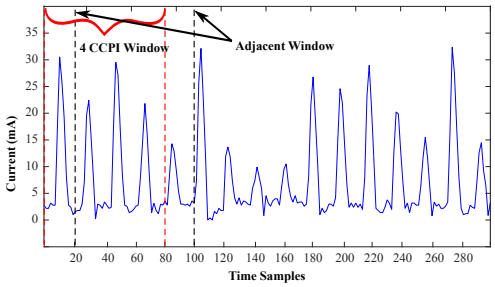


Fig. 5. Sliding-Window scheme for a 4-CCPI where the next subsequent window is shifted by 1-clock-cycle

1) CCPI Windowing: For classification, a known sequence of instructions is executed over a continuous loop while side-channel leakage measurements are collected. Adjoining clock-cycles in each waveform are combined to form N-cycle windows. As a result, subsequent windows within a waveform capture appear after a fixed number of time samples equivalent to that required for a single clock cycle. A N-cycle window may vary in size from one 1-CCPI (minimum CCPI required for a register access addressing mode instruction) to six 6-CCPI (maximum CC required for an indexed addressing mode instruction). For a specific window size N, an iterative shift by 1-clock-cycle, over the span of the waveform, generates the templates (N-CCPI instruction) for a dictionary $\mathbf{C}_N$. For example, Fig. 5 shows a 4-CCPI window and an adjacent window. All of the data captures acquired over the monitoring instrument are retained for analysis.

2) Classifying CCPI: We present an algorithm to determine a likely order of CCPI in an instruction sequence based on an optimal fit of instruction templates to build a matched waveform to the observed data. The observed test data for a sequence is defined to be the average power-supply transient of

10 000 observations of the sequence execution. This decision involves solving two problems. The first is deciding the optimal tagging of instruction-initiating-clock-cycles in a clock-cycle sequence c_1, c_2, ..., c_n. For any solution we assume that we know that an instruction starts in the first clock-cycle, so c_1 is always an instruction-initiating-clock-cycle. As an example, for a two-clock-cycle sequence choice, a choice for the set for labeling as instruction-initiating-clock-cycles could be $\{c_1, c_2\}$, meaning that there are two instructions, starting at clock-cycle 1 and clock-cycle 2 and are each 1-clock-cycle instructions. The alternative choice is $\{c_1\}$, meaning there is only a single 2-CCPI. For a 3-CCPI observation the choice could be $\{c_1\}$, $\{c_1, c_2\}$, $\{c_1, c_2, c_3\}$, or $\{c_1, c_3\}$. These decisions correspond to CCPI sequences of (3), $(1, 2)$, $(1, 1, 1)$ and $(2, 1)$. In general, the search space for the optimal tagging of instruction-initiating-clock-cycles is of size $2^{\mathcal{K}-1}-1$ where $\mathcal{K}$ is the number of observed instruction clock-cycles. It can be reasoned that determining the instruction-initiating-clock-cycles is the same as determining an N-CCPI sequence.

The second problem is in determining the cost function for deciding the optimal labeling of instruction-initiating-clock-cycles, and thus the sequence of clock-cycle lengths, $\vec{L} = (L_1, ..., L_k)$. For each entry L_j, in the series $\vec{L}$, a clock-cycle offset $\mathcal{O}_k$ can be determined from the summation of the previous values, $\sum_{n=1}^{j-1} L_n$. Taking k to be the number of power supply waveform samples taken per clock-cycle, the cost of each label is the minimum of the total squared distance between any template waveform in our template book that comprises $N \times k$ samples and the observed power supply waveform over the samples $\mathcal{O}_j \times k + 1$ to $(\mathcal{O}_j + L_j) \times k$. In other words, for every possible span of clock-cycles we take minimum cost for identifying that span as a single contiguous instruction to be the squared sum distance of that power supply waveform span to the best matching template function in our template book. In addition to the raw power profile waveform, we also utilize PCA based templates when the minimum of total squared distances are calculated. We precompute and label the solutions to these minimal cost computations as $m_{\mathcal{O},\mathcal{N}}$, which denote the cost of deciding there is an instruction of clock-cycle-length N at clock-cycle offset $\mathcal{O}$.

$$M_{\mathcal{O},N} = \min_{\forall i \in 1..\text{length}(C_N)} \sqrt{\sum_{n=0}^{k \times l} (T_{N,i}[n] - D[\mathcal{O} \times N + n])^2} \tag{2}$$

where k is the number of power supply data points taken in an clock-cycle; $\mathbf{C}_N$ is a book of template waveforms for each instruction that takes N clock-cycles; $T_{N,i}[n]$ is data point n of the i^{th} template in $\vec{C}_N$; $D[n]$ is data point n in the captured power supply waveform; and thus $M_{\mathcal{O},N}$ represents the minimum Euclidean distance for having an instruction with clock-cycle offset $\mathcal{O}$ and clock-cycle-length N. With these precomputed results, we can turn to a dynamic programming solution to decide the optimal sequence of clock-cycle lengths.

Starting at the first position, we are challenged to decide the optimal choice of instruction length N and thus the second instruction-initiating-cycle label. The cost of a given decision for N is the summation of the cost $M_{1,N}$ plus the cost of the remaining optimally decided labels chosen from clock-cycle c_{N+1} onward that includes the instruction-initiating-cycle labeling on c_{N+1}. Thus each decision introduces the subproblem of computing the remaining labels and their cost, which represents a solution to an *optimal-substructure* [22] to find the best overall solution. The number of choices for $M_{2,N}$ depend on the size of the set of possible clock-cycle-lengths. The decision with the lowest sum cost is chosen. Among the $2^{\mathcal{K}-1}-1$ possible choices sets of instruction-initiating-clock-cycles, a given clock-cycle may be labeled as an instruction-initiating-clock-cycle, c_j in multiple possible sets. A labeling of c_j represents a subproblem of an *optimal-substructure* solution beginning at that clock-cycle. Therefore, we have *overlapping subproblems* [22] in choosing between overall solutions, making the overall problem solution appropriate for dynamic programming. For a given clock-cycle offset, there is a minimum cost (sum squared error, DPCA^2 in (1)) in representing the remaining waveform using template waveforms in our dictionary. This minimum cost is denoted $C_{\mathcal{O}}$. Let $\mathcal{K}$ be the total number of clock-cycles in the overall observed sequence.

Algorithm 1 Optimal CCPI order

```
1:  procedure DETERMINEOPTIMALCCPI(M)
2:      for i = 1 to L_seq do   ▷ Initialization of cost matrix
3:          for j = 1 to i − 1 do
4:              C_{i,j} = ∞
5:          j = i
6:          while j < L_seq and j < L_win + i do
7:              C_{i,j} = M_{i,(j−i+1)}
8:              j = j + 1
9:          while j < L_seq do
10:             C_{i,j} = ∞
11:             j = j + 1
12:     for l = 2 to L_seq do          ▷ Dynamic programming
13:         for i = 1 to L_seq − l + 1 do
14:             j = i + l − 1
15:             for s = i to j − 1 do
16:                 if C_{i,s} + C_{s+1,j} < C_{i,j} then
17:                     C_{i,j} = C_{i,s} + C_{s+1,j}
18:                     split_{i,j} = s          ▷ For recovery
19:     optimalCCPI_1 = split_{1,L_seq}  ▷ Recover optimal path
20:     index = 2
21:     i = split_{1,L_seq} + 1
22:     psplit = split_{1,L_seq}
23:     while split_{i,L_seq} ≠ 0 do
24:         optimalCCPI_index = split_{i,L_seq} − psplit
25:         psplit = split_{i,L_seq}
26:         i = split_{i,L_seq} + 1
27:         index = index + 1
28:     optimalCCPI_index = L_seq − psplit
```

Then, the subproblem may be solved recursively as

$$C_{\mathcal{O}} = \begin{cases} \min_{\forall \mathcal{N} \in 1..\mathcal{K}-\mathcal{O}+1} M_{\mathcal{O},N} + C_{\mathcal{O}+N} & \mathcal{O} \leq \mathcal{K} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The recurrence relation in equation 3 for $C_{\mathcal{O}}$ allows for *memoization* of optimal solutions to subproblems $C_{\mathcal{O}+N}$, where $C_{\mathcal{O}}$ *remembers* the most current optimal solution to

a subproblem. The optimal solution is available at $C_{\mathcal{K}}$. For readability, clock-cycle offsets $\mathcal{O}_i$, $\mathcal{O}_j$ and $\mathcal{O}_l$, are represented by i, j and l respectively. $split$ and $psplit$ are temporary placeholders where s holds the indexes for *substructures* utilized during the memoization step to determine the optimal solution. L_{seq} is the clock-cycle length for the instruction sequence being observed. L_{win} is the maximum clock-cycle window length possible for the current instruction set. The optimal sequence of CCPI is available from $optimalCCPI$.

3) Optimal CCPI Algorithm: Algorithm 1, determines the optimal sequence of CCPI for an instruction sequence by evaluating the recurrence relation defined in equation 3.

IV. Evaluation

In this section, we demonstrate the effectiveness of our technique w.r.t. the CCPI reported in the MSP430 manual. Our technique determines the optimal CCPI order across a number of random sequences of instructions. Although we have extensive data from multiple random as well as predefined sequences (e.g. AES, PID controller etc.), for the purpose of brevity, given the constraints of this paper, two sample sequences are listed in tables II and III with the correct predictions marked in blue. Prediction results from N-CCPI PCA templates constructed from individual power supply pin data, majority decision from these individual results (MV) and composite template (Comb.) are presented.

TABLE II
Optimal order of CCPI (Sequence 1)

Sequence 1	CCPI	Optimal Clock-Cycle Sequences										
		Power Pin				MV	Comb.	Power Sum	CR1	CR2	CR3	CR4
		1	2	3	4							
pop_mem_reg	2	1,1	1,1	1,1	1,1	1,1	1,1	1,1	87.76	89.98	93.34	88.84
add_mem_mem_nosub	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	84.50	83.94	87.38	74.74
inc_reg_const_ind_nosub	4	**4**	**4**	**4**	1,3	**4**	**4**	**4**	99.72	99.44	99.48	99.74
mov_mem_ind_nosub	5	**5**	6	**5**	6	-	6	**5**	97.96	97.84	96.48	97.50
add_reg_reg	1	**1**		5		-		**1**	98.66	98.66	98.66	98.68
sub_mem_mem_sub	6	**6**	**6**	3	**6**	**6**	**6**	**6**	91.94	86.52	87.96	82.18
dec_const_reg	1	**1**	**1**		**1**	**1**	**1**	**1**	98.70	98.70	98.66	98.68
mov_ind_reg_nosub	3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	99.98	100.00	99.76	99.96
subc_imm_reg_sub	2	1,1	1,1	1,1	1,1	1,1	1,1	1,1	95.38	99.46	98.88	95.06
bit_mem_mem_nosub	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	96.22	96.60	96.54	97.44
cmp_mem_mem_sub	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	95.28	97.12	98.14	98.42
xor_reg_const_ind_nosub	4	**4**	**4**	5	5	-	**5**	**4**	56.36	46.30	66.60	56.18
inc_const_reg	1	**1**	**1**			-		**1**	98.82	98.82	98.74	98.74
Total	45	45	45	45	45		45	45	92.41	80.50	93.89	91.24

TABLE III
Optimal order of CCPI (Sequence 2)

Sequence 2	CCPI	Optimal Clock-Cycle Sequences										
		Power Pin				MV	Comb.	Power Sum	CR1	CR2	CR3	CR4
		1	2	3	4							
pop_mem_reg	2	1,1	1,1	6	6	-	1,1	1,1	98.16	98.30	98.48	96.76
inc_reg_const_ind_nosub	4	**4**	**4**			-	**4**	**4**	99.00	97.80	99.28	99.02
mov_mem_ind_nosub	5	**5**	6	**5**	6	-	6	**5**	90.72	93.32	91.02	88.34
add_reg_reg	1	**1**				-		**1**	95.44	95.44	95.44	95.46
sub_mem_mem_sub	6	**6**	**6**	5,3	**6**	**6**	**6**	**6**	95.98	93.20	89.54	86.84
dec_const_reg	1	**1**	**1**		**1**	**1**	**1**	**1**	95.40	95.64	95.82	95.46
add_mem_mem_nosub	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	88.66	88.98	86.46	86.76
dec_const_reg	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	96.74	97.62	96.50	97.10
mov_ind_reg_nosub	3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	96.08	95.76	95.12	95.90
subc_imm_reg_sub	2	1,1	1,1	1,1	1,1	1,1	1,1	1,1	92.16	95.28	94.88	92.74
bit_mem_mem_nosub	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	97.18	98.24	97.86	99.44
cmp_mem_mem_sub	5	4	4	4	4	4	4	4	96.88	97.94	98.54	97.30
xor_reg_const_ind_nosub	4	5	6	5	5	5	6	5	64.18	47.88	74.88	65.66
inc_const_reg	1	**1**		**1**		-		**1**	99.86	99.86	99.88	99.88
Total	46	46	46	46	46		46	46	93.32	89.96	93.84	92.62

CR values 1-4 (for templates from power pins 1-4 respectively) depict the classification rates once the clock cycles order for a specific sequence has been determined which is the final step in the instruction sequence recovery process. The rates of classification vary between 80.5% and 93.89%. For some entries in the table (entry '1,1'), the clock cycles predicted have been aligned to match the expected clock sequence. In addition, we determine $\mathcal{P}_{sum}[n] = \sum_{i=1}^{Npad} \mathcal{P}_i[n]$, where $Npad$ (4) is the number of power supply pins. $\mathcal{P}_i[n]$ is the power data from power supply pin i at time sample n, $\mathcal{P}_{sum}[n]$ is the power data sum result ("Power Sum") from all power supply pins at time sample n. $\mathcal{P}_{sum}$ is calculated prior to the template generation step which was summarized in §III. We find that the results for "Power Sum" give the best predictions on the order of CCPI over all the random sequences of instructions.

Most often, 2-clock-cycle instructions are mis-predicted as two 1-clock-cycle instructions (Table IV). This can primarily be attributed to the two pipeline stage architecture of openMSP430. The "decode and fetch" stage of any subsequent instruction is in the pipeline with the last "execute" clock-cycle of an instruction that is currently being executed. For 2-clock-cycle instructions, this translates to a higher power consumption during the second clock-cycle of the "execute" stage. Furthermore, if a 1-clock-cycle instruction is in its "execute" stage, the "decode and fetch" stage for the next instruction is in pipeline.

TABLE IV
Prediction rates per Instruction

CCPI	Instructions	CCPI Prediction Rates (%)					Freq. %
		Pin 1	Pin 2	Pin 3	Pin 4	Power Sum	
2	pop_mem_reg	0.00	0.00	0.00	0.00	0.00	7.52
5	add_mem_mem_nosub	100.00	100.00	90.91	100.00	100.00	8.27
4	inc_reg_const_ind_nosub	100.00	100.00	62.50	75.00	100.00	6.01
5	mov_mem_ind_nosub	100.00	12.50	75.00	12.50	100.00	6.01
1	add_reg_reg	90.00	20.00	10.00	20.00	100.00	7.52
6	sub_mem_mem_sub	90.91	100.00	45.45	100.00	100.00	8.27
1	dec_const_reg	100.00	100.00	62.50	100.00	100.00	12.03
3	mov_ind_reg_nosub	90.00	100.00	90.00	80.00	100.00	7.52
2	subc_imm_reg_sub	0.00	0.00	0.00	22.22	0.00	6.77
6	bit_mem_mem_nosub	100.00	100.00	88.89	100.00	100.00	6.77
5	cmp_mem_mem_sub	72.73	72.73	72.73	72.73	72.73	8.27
4	xor_reg_const_ind_nosub	36.36	36.36	18.18	18.18	45.45	8.27
1	inc_const_reg	25.00	12.50	37.50	0.00	75.00	6.01
1	nop_reg_reg	0.00	100.00	100.00	0.00	100.00	0.76
	Overall Prediction Rate (%)	**70.68**	**61.65**	**51.13**	**57.14**	**77.44**	

TABLE V
Prediction rates per Instruction based on selection from combined codebook of templates

CCPI	Instructions	CCPI Prediction Rates (%)					Freq. %
		Pin 1	Pin 2	Pin 3	Pin 4	Power Sum	
2	pop_mem_reg	100.00	100.00	90.00	100.00	100.00	7.52
5	add_mem_mem_nosub	100.00	100.00	100.00	100.00	100.00	8.27
4	inc_reg_const_ind_nosub	100.00	100.00	100.00	100.00	100.00	6.01
5	mov_mem_ind_nosub	100.00	100.00	87.50	100.00	100.00	6.01
1	add_reg_reg	100.00	100.00	80.00	100.00	100.00	7.52
6	sub_mem_mem_sub	100.00	100.00	100.00	100.00	100.00	8.27
1	dec_const_reg	100.00	100.00	100.00	100.00	100.00	12.03
3	mov_ind_reg_nosub	100.00	100.00	100.00	100.00	100.00	7.52
2	subc_imm_reg_sub	100.00	100.00	100.00	100.00	100.00	6.77
6	bit_mem_mem_nosub	100.00	100.00	100.00	100.00	100.00	6.77
5	cmp_mem_mem_sub	100.00	100.00	81.81	100.00	100.00	8.27
4	xor_reg_const_ind_nosub	100.00	100.00	36.36	100.00	100.00	8.27
1	inc_const_reg	100.00	100.00	12.50	100.00	100.00	6.01
1	nop_reg_reg	100.00	100.00	100.00	100.00	100.00	0.76
	Overall Prediction Rate (%)	**100.00**	**100.00**	**84.96**	**100.00**	**100.00**	

The two 1-clock-cycle templates tends to be similar to two execute clock-cycles of a 2-clock-cycle instruction as opposed to the 2-CCPI templates for 2-clock-cycle instructions. A similar argument can be made for the low predictability of the 4-clock-cycle "xor" instruction followed by a a 1-clock-cycle "inc" instruction.

To improve CCPI sequence recovery rates, we re-train templates (as outlined in §III), such that, an instruction that

succeeds an observed instruction is, either one that performs a register fetch, or, one that accesses the data memory. As a result, we have two dictionaries $\mathbf{CR}_N$ and $\mathbf{CM}_N$ which hold template waveforms for each instruction that takes N clock-cycles with a succeeding instruction that performs a register fetch and an instruction that perform a memory access respectively. As a result, $TR_{N,i}[n]$ is data point n of the i^{th} template in $\vec{CR}_N$ and $TM_{N,i}[n]$ is data point n of the i^{th} template in $\vec{CM}_N$. Hence we find the minimal euclidean distances $MR_{\mathcal{O},N}$ and $MM_{\mathcal{O},N}$ and construct $M'_{\mathcal{O},N} = min(MR_{\mathcal{O},N}, MM_{\mathcal{O},N})\ \forall\ \mathcal{O}, N$. Thus, the recurrence equation 3, can be modified to accommodate the multiple dictionaries to create a combined codebook of templates

$$C_{\mathcal{O}} = \begin{cases} \min\limits_{\forall \mathcal{N} \in 1..\mathcal{K}-\mathcal{O}+1} M'_{\mathcal{O},N} + C_{\mathcal{O}+N} & \mathcal{O} \leq \mathcal{K} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

As Table V shows, the improvement using a combined codebook of templates allow for 100 % prediction rates per instruction from individual power supply pins.

A. Feasibility of CCPI Sequence Determination

For the following case study, results presented in tables II and III demonstrate feasibility of determining CCPI order. Sequence 1 was an untampered sequence of instructions (contains only one *'dec'* instruction). Sequence 2 comprises a re-ordered sequence of instructions with a second *'dec'* instruction introduced after instruction seven. The modified sequence is immediately obvious when comparing the reported CCPI order for the two sequences. The hardware-utilization-based classification that follows the CCPI order discovery serves to further mitigate any uncertainties in actual classes the instructions belong to. To this effect, our technique covers a wide range of anomalies, ranging from code insertions, and re-arrangement of instructions in a code sequence where the order of CCPI in a code sequence is affected. As a part of our analysis we could recover 94% of CCPI order in an AES encryption loop (1679 instructions - 4231 clock cycles) and 97% of CCPI order in a PID controller sequence (58 instructions - 177 clock cycles). We tested our hypothesis on anomaly detection with the PID controller code by injecting malicious code which would modify the setpoint values after the derivative gain is applied. We could distinctly identify the modified code sequence from the untampered controller code.

In our analysis of several sample sets of random sequences, CCPI order recovery rates ranged from 69.2% to 87.5% using the optimalCCPI order algorithm on the templates constructed using the "Power Sum" power profile data using a single codebook of templates. With combined codebook of templates, the prediction rates are improved to 100%. These recovery rates are significantly higher than the recovery rates reported in [14]. Furthermore, the application of this technique is a first for microcontrollers with many-clock-cycle instructions.

V. Conclusion

This work presents a novel non-intrusive technique to detect multi-clock-cycle instruction sequences on a pipelined architecture, based on the analysis of power-supply transients. Our technique uses PCA augmented with a dynamic programming algorithm that uses observations from multiple power supply pins. With 10 000 observations of a execution sequence, our technique achieved a classification accuracy between 87% to 100% using just 10 PCA components. Using multiple dictionaries, we determined the CCPI sequence with 100 % accuracy. As demonstrated in §IV, our technique can determine tampering of firmware running on a general purpose pipelined embedded platform.

References

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO '99*. Springer-Verlag, 1999, pp. 388–397.

[2] M. Banga and M. Hsiao, "A region based approach for the identification of hardware trojans," in *HOST 2008. IEEE International Workshop on*, June 2008, pp. 40–47.

[3] P. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *CRYPTO '96*. Springer Berlin Heidelberg, 1996, vol. 1109, pp. 104–113.

[4] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*. Springer Berlin Heidelberg, 2001, vol. 2140, pp. 200–210.

[5] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side-channel(s)," in *CHES 2003*. Springer-Verlag, 2003, pp. 29–45.

[6] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *CRYPTO '97*. Springer Berlin Heidelberg, 1997, vol. 1294, pp. 513–525.

[7] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," in *ITC 2004*, Oct 2004, pp. 339–344.

[8] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel," *IACR Cryptology ePrint Archive*, vol. 2002, p. 169, 2002.

[9] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of des implemented on computers with cache," in *CHES 2003*. Springer Berlin Heidelberg, 2003, vol. 2779, pp. 62–76.

[10] M. Kuhn, "Cipher instruction search attack on the bus-encryption security microcontroller ds5002fp," *Computers, IEEE Transactions on*, vol. 47, no. 10, pp. 1153–1157, Oct 1998.

[11] T. Gilmont, J. Legat, and J.-J. Quisquater, "Enhancing security in the memory management unit," in *EUROMICRO Conference, 1999. Proceedings. 25th*, vol. 1, 1999, pp. 449–456 vol.1.

[12] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *2004 IEEE Symposium on Security and Privacy, Berkeley, CA, USA*, 2004, pp. 3–11.

[13] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *CRYPTO 2014*, 2014, pp. 444–461.

[14] T. Eisenbarth, C. Paar, and B. Weghenkel, *Transactions on Computational Science X: Special Issue on Security in Computing, Part I*. Springer Berlin Heidelberg, 2010, ch. Building a Side Channel Based Disassembler, pp. 78–99.

[15] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs," in *CHES 2014*. Springer Berlin Heidelberg, 2014, vol. 8731, pp. 242–260.

[16] M. Msgna, K. Markantonakis, and K. Mayes, *ISPEC 2014*. Springer International Publishing, 2014, ch. Precise Instruction-Level Side Channel Profiling of Embedded Processors, pp. 129–143.

[17] M. Msgna, K. Markantonakis, D. Naccache, and K. Mayes, *COSADE 2014*. Springer International Publishing, 2014, ch. Verifying Software Integrity in Embedded Systems: A Side Channel Approach, pp. 261–280.

[18] R. Callan, A. Zajić, and M. Prvulovic, "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*. IEEE Computer Society, 2014, pp. 242–254.

[19] O. Girard, "openmsp430," http://opencores.org/project,openmsp430, 2016 (accessed March 1, 2016).

[20] *MSP430x1xx Family User's Guide*, (rev. f) ed., Texas Instruments, http://www.ti.com/lit/ug/slau049f/slau049f.pdf, 2006.

[21] I. Jolliffe, *Principal component analysis*. Wiley Online Library.

[22] T. T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,Third Edition*. MIT Press, 1990.