An Efficient Profiling Attack to Real Codes of PIC16F690 and ARM Cortex-M3

SHAHRAM VAFA1, MASSOUD MASOUMI2, AND AMIR AMINI1
1Department of Electrical Engineering, College of Technical and Engineering, Islamic Azad University, West Tehran Branch, Tehran 1468763785, Iran
2Department of Electrical Engineering, Islamic Azad University, Islamshahr Branch, Tehran 33147-67653, Iran
Corresponding author: Massoud Masoumi (massoud_masoumi@yahoo.com)

ABSTRACT This article presents a new and efficient method based on power analysis, hierarchical recognition of instructions, and machine learning for reverse engineering of the instructions of PIC16F690 as an 8-bit microcontroller and LPC1768, which includes an ARM Cortex-M3 core as a 32-bit platform. Both dynamic and static power consumption were considered and analyzed. The instructions were classified in different Hamming weight groups using ensemble classification algorithms along with the Kullback-Leibler feature selection method to improve the recognition rate of opcodes and operands of real instructions. Results demonstrated 99.5% and 93.3% average success rate in recovering test instructions and real codes of PIC16F690, respectively. This work also presents promising results in reverse engineering of the instructions of LPC1768 with an overall recognition rate of 98% for test codes and 80.2% for real codes. To the best of our knowledge, this is the first serious report about profiling attack to a 32-bit platform without the need for any sophisticated laboratory tools.

INDEX TERMS Template attacks, reverse engineering of instructions, ensemble classification.

I. INTRODUCTION

Side channel attacks are hardware cryptanalytic attacks that can pose efficient and serious threat to various systems from small security tokens to industrial control systems and critical infrastructures [1]–[3]. The notorious Meltdown and Spectre attacks that can affect CPUs even eighth-generation core platforms, are clear examples. However, these attacks involve only software and mainly rely on speculative execution, which helps speed up the execution. The so-called profiling attacks are a subcategory of power analysis attacks that are used for code breaking and reverse engineering purposes. They enable attackers to not only reveal secret keys of a particular device but also to retrieve its secret program codes or algorithm without having any prior knowledge about the running algorithm on the target device. These attacks are one of the most powerful and dangerous kinds of side-channel attacks that consist of two major phases. First, the adversary procures a copy of the target platform and tries to find any meaningful dependency between data obtained from the target device and device operation. Then, he mounts a code or key-recovery

The associate editor coordinating the review of this manuscript and approving it for publication was Yanjiao Chen.

NOMENCLATURE

AES Advanced encryption standard
KL Kullback-Leibler
KLD Kullback-Leibler Divergence
ALU Arithmetic Logic Unit
KNN k-Nearest Neighbours
CWT Continuous wavelet transform
LDA Linear discriminant analysis
DPA Dynamic power analysis
ML Machine learning
ECDSA Elliptic curve digital signature algorithm
PCA Principle Component Analysis
EM Electromagnetic emission
QDA Quadratic discriminant analysis
FN False Negative
RF Random forest
FP False Positive
SCA Side-channel analysis, or side-channel attack
HD Hamming distance
SPA Static power analysis
HW Hamming weight
SVM Support vector machine

IoT Internet of Things
TA Template attack
IR Instruction register
TN True Negative
attacking on the victim device. Profiling attacks mainly include template attacks and stochastic cryptanalysis. Using such attacks, an attacker can extract the cipher key and use it to make adverse configuration changes, cause the system to malfunction or have it send false data back to the master system, which can have serious consequences.

There are many papers about template and profiling attacks and instruction disassembling. Most of the existing papers deal with 8-bit processors with relatively simple architectures such as Microchip PICs or AVR s. This is mainly because these target platforms are easy to have access to and to use for related experiments. In addition, they mostly evaluate their methods on test codes, while evaluating the efficiency of the attack in the breaking of actual codes running in a real program is much more important. Although template attacks have shown their effectiveness in recent years, they still suffer from numerical problems and need some assumptions regarding high data dimensionality and noise distribution, which limit their range of applications. Usually, they rely on a large amount of accurate and high quality profiling traces, which makes them almost impractical in many cases, specifically when there are only a limited number of power traces with low signal-to-noise ratio. Machine learning (ML) based methods can prevent these problems, and are found to perform better than previously known techniques when faced with complexity in numerical and statistical calculations or some of the restrictive assumptions on noise distribution [4]–[15]. Power analysis attacks compromise the security of cryptographic devices through analyzing their power consumption. Profiling attacks use a more effective statistical modelling of power consumption in such a way that they estimate the conditional probability density function of the time series for each possible key-dependent information through a Gaussian parametric model. ML-based techniques do not require any parametric or normal assumption. Based on these facts, this work presents a different method from what has been presented in the literature, and achieves far better results compared with other works published in other papers. By combining machine-learning-based methods with an efficient hierarchical trace classification approach and instruction grouping, we have presented a powerful side-channel-based disassembler that is able to reveal the instructions of PIC16F690 and ARM Cortex-M3 with 99.5% and 98% average success rate, respectively. We have chosen PIC16F690 to have a fair comparison with other existing works [8], [10], [14], and ARM Cortex-M3 as a 32-bit low power and relatively high-performance target with more complex architecture. To the best of our knowledge, this is the first serious report about profiling attack to 32-bit platforms. We used the Advanced Encryption Standard (AES) as the underlying algorithm for evaluating the efficiency of the attack on real codes in both target devices.

In this work, we demonstrate how to overcome some previously identified shortcomings of power analysis-based profiling attacks using the ML techniques, which not only improves the solutions but improves the accuracy of templates and computations as well. We demonstrated how to use ML techniques as a powerful alternative to standard side-channel evaluation methods. In addition, unlike other papers that focus only on dynamic power analysis, we have analyzed both dynamic and static power analysis in order to make our method more powerful than other previous works. To explain, unlike other works, we are able to recover actual codes of both processors, not their test codes. To the best of our knowledge, this is the first serious report about a successful profiling attack, specifically an ML-based profiling attack to real codes of a 32-bit target platform. Therefore, the main contributions of this work can be summarized as follows:

- Performing both dynamic and static power analyses to extract the maximum possible information leakage from the power consumption of the target devices in such a way that dynamic power analysis is used to reveal the Hamming weight (HW) of each instruction’s operand, while using static power analysis to recover the total HW of the opcode and operand of the next instruction.

- Adopting instruction grouping and hierarchal instruction classification methods to reduce the number of required traces and computational costs for mounting the attack.

- Employing t-distributed Stochastic Neighbor Embedding (t-SNE) machine learning algorithm for instruction grouping and Kullback-Liebler (KL) feature selection and Principle Component Analysis (PCA) for dimension reduction.

- Using ensemble of classifiers method to improve the classification rate of instructions.

- Achieving 99.5% average success rate for test codes and 93.3% for real codes in reverse engineering of the most commonly used instructions of 8-bit PIC16F690 microcontroller.

- Achieving 98% average success rate for test codes and 80.2% for real codes of LPC1768, which includes a 32-bit Cortex-M3 microcontroller for embedded applications.

This article is organized as follows: in sections II some previous works are reviewed. Section III presents a brief overview on PIC16F690 and LPC1768. Hierarchal instruction recognition is discussed in section IV. Power consumption of PIC16F690 and pipelined architectures are discussed in section V. Instruction grouping and classification using static and dynamic power traces are briefly illustrated in section VI. Section VII presents machine-learning-based instruction recognition. In section VIII, the results of the implementation of the attack against the target processors are presented. The scalability and computational cost of the proposed approach are discussed in section IX. This work is compared with other works in section X. Finally, in the conclusions, we summarize our results and discussions.

II. PREVIOUS WORKS

In [4], Choudary and Kuhn present a so-called portable template attack that is able to recover key bytes from a hardware implementation of the Advanced Encryption Standard on an ATMEL AVR XMEGA 8-bit microcontroller with 85% success rate. They present several methods to reduce numerical problems that happen during the statistical calculations.
related to a practical template attack. The results are valuable and interesting; however, they do not present comprehensive results for reverse engineering of the instructions. Instead, they focus on a single ‘LOAD’ instruction and use it to break the cipher while transferring the cipher key on data bus. Their method is not based on intelligent or ML algorithms. In addition, they do not provide any results about 16 or 32-bit target platforms. Eisenbarth et al. in [8] mounted power-analysis-based template attack on PIC16F687 and achieved 70% recognition rate for test instructions and 58% for real instructions by exploiting prior knowledge about its program code through hidden Markov model and Viterbi algorithm. Msgrna et al. [9] classified power traces of instructions of ATMega163 using PCA and k-Nearest Neighbors (k-NN) algorithms. They achieved 100% recognition rate for test codes, but they were not able to reproduce the results for real codes. In their article, Fisher’s Linear Discriminant Analysis (LDA) in combination with k-NN achieves 48.74% recognition rate. While PCA with k-NN achieves a slightly better result of 56.88%, plain PCA with k-NN unexpectedly improves the recognition rate up to 100%. Strobel et al. in [10] raised the recognition rate to 88% for real codes running on a PIC16F687 via EM analysis with multiple antennas. Nevertheless, their method needs decapsulating facilities with several extra processes which increases the complexity and cost of the attack. D-Tsague and Twala in [11] measured EM emanations of ATMega163 on a smart card and achieved 78.3% classification success rate by combining PCA and k-NN algorithms. Park et al. in [12] presented a side-channel-based disassembler using QDA and SVM for data transfer instructions in ATMega328p AVR microcontroller that is able to identify test instructions with almost 99.03% accuracy. However, they did not evaluate their attack on real codes and left it to the future. They published a comprehensive version of their work in [13] with almost the same results. Cristiani et al. in [14] proposed a new approach for side-channel-based disassembler that directly focuses on bit encoding of an instruction using local EM leakage. They employed high-precision motorized XYZ for EM probe and PIC16F15376 as the target device to build a bit-level classifier, and achieved 99.41% recognition rate at bit level and 95% at the full 14-bits instructions. However, like most of the other papers, they left to the future the evaluation of their approach on pipelined and more complex processors. Medwed and Oswald in [16] present a practical template attack against the implementation of elliptic curve digital signature algorithm (ECDSA) on ARM-7 processor, but they do not provide any detail about the efficiency of the attack, the success rate or the number of required traces. McCann et al. [17] have used instruction-level power models for ARM Cortex M0 to emulate leakages and to detect even subtle leakages in the implementations. The results are interesting but their work is different with the traditional power side-channel models that estimate secret data as they present a circuit-level simulator based on data dependent switching effects for ARM. Also, they do not present an actual disassembler for code breaking or reverse engineering.

III. PIC16F690 AND LPC1768

In the PIC architecture, all instructions are executed within a single instruction cycle, unless a conditional test or branching occurs, or the program counter is changed as a result of the execution of an instruction. In such cases, the execution takes two instruction cycles. Such performance is mainly caused by two-stage pipeline which means that, while one instruction is decoded and executed, the next cycle is executed as NOP, where NOP stands for no operation with zero HW, and has the lowest effect on power consumption of other instructions. Each instruction cycle consists of four oscillator periods referred to as Q1 to Q4. For an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1μs.

The LPC1768 includes an ARM Cortex-M3 core and operates at CPU frequencies of up to 100 MHz. The Cortex-M3 includes a three-stage pipeline of fetch, decode and execute, and uses a Harvard architecture with separate local instructions and data buses along with a third bus for peripherals. It also contains an internal pre-fetch unit that supports speculative branching. Cortex M-3 has 16 32-bit registers, three of which are reserved for stack pointer, program counter, and link register. ARM processors have two different instruction sets: ARM instructions and Thumb instructions. ARM instructions are 32-bit instructions while Thumb instructions are 16-bit instructions extended with Thumb-2 32-bit instructions. Cortex-M3 does not support ARM instructions and implements ARMv7-M Thumb instruction set. It supports a variable-length instruction set that provides both 32-bit and 16-bit instructions for improved code density. On execution, 16-bit Thumb instructions are decompressed and decoded to full 32-bit ARM instructions, without performance loss [18]. In this work, we mainly focus on a selected number of 16-bit Thumb instructions that are mostly used in the implementation of symmetric cryptographic operations. The selected Thumb instructions are those mainly employed for data processing, shifting, and data transfer: ‘LSL’, ‘AND’, ‘TST’, ‘ADC’, ‘EOR’, ‘ASR’, ‘MVNS’, ‘SUBS’, ‘STR’, ‘LDR’, ‘ARD’, ‘ADDS’, ‘MOVS’ and ‘CMP’. Many of the other instructions that exist in the Thumb instruction set such as loading or storing a half word also fall into similar or the same categories.

IV. HIERARCHICAL INSTRUCTION RECOGNITION

In ML-based instruction disassembling, power consumption features of target instructions are collected to form feature vector for ML prediction. A hierarchical approach maps ML computations to a corresponding hierarchical order to reduce computational cost and increase the accuracy. Hierarchical classification intrinsically requires to execute $O(N\log N)$ cascaded binary classifiers for an $N$ dimensional C-classes classification problem, which is lower than other well-known
binary classifiers such as one-vs-all and one-vs-one as shown in Table 1 [19]. The process of instruction disassembling for a typical microcontroller is usually performed hierarchically in four different steps as demonstrated in Fig. 1. At first, feature selection and dimension reduction are used as preprocessing steps for mapping high-dimensional data to a lower-dimensional space. Measured power traces are then clustered into instruction groups based on their operation in the second stage. In the third step, instructions in each selected group are classified based on their HWs, and finally each instruction is recognized by identifying the HW of its opcode and operand.

| Learning        | Memory | Computational Complexity |
|-----------------|--------|-------------------------|
| One-vs-all      | NC     | O(NC)                   |
| One-vs-one      | N(C − 1)/2 | O((N(C − 1)/2) |
| Hierarchical    | N(C − 1) | O(N log₂ N) |

V. INSTRUCTION POWER ANALYSIS IN PIC AND ARM

In PIC processors, the rate of the clock significantly affects the shape of power consumption traces, mainly due to the charging and discharging of internal capacitances. Moreover, the fetching process influences the power consumption of Q2 to Q4 proportionally to the HW of fetched instructions. In the first oscillator cycle or Q1, the fetched instruction is stored in the instruction register (IR) so that in case of an arithmetic or logical instruction, the operand is sent to the ALU through the ALU multiplexer. In case of a file-register operation, the operand is sent from the SRAM memory to the ALU through the same multiplexer. After that, the previous data stored on the bus is replaced by the operand of the instruction. This leads to a power consumption that is proportional to the Hamming distance (HD) of both values. In Q2, the ALU reads the operand on the data bus. Hence, the HW of the opcodes is the most important factor that affects the power consumption, specifically in the case of the literal or file-register operations in which their address are stored in the opcodes. The ALU processes the instructions in Q3 and the result is transferred to the data bus. Since the actual value on data bus is the operand loaded in Q1, the power consumption directly depends on the HD of these values. Finally, in the last oscillator cycle or Q4, the result on the data bus is stored in memory or any external device connected to the data bus. It should also be mentioned that the next instruction is latched into instruction register in Q4 cycle causing the power consumption of the device to be directly proportional to the HD of the executed and the latched instructions [20].

In a pipelined architecture, the power consumed in one clock cycle is the sum of the power consumed in all pipeline stages at the same clock cycle. This means that the power consumption of each instruction includes the power consumption of the same instruction and inter-instruction effects due to the existence of multiple instructions inside the pipeline. Thus, we need to consider changes caused by instructions in different pipeline stages and power consumption of each instruction including the power consumption of the same instruction and inter-instruction effects due to the existence of other instructions in the pipeline [21]–[23]. Therefore, power consumption of each instruction in clock cycle n can be described by Eq. (1).

\[ P_{\text{Cycle}}(n) = \sum_{s=1}^{M} (P_b(I_s) + P_{I_{s-1}, I_s}) \]

In Eq. (1), \( M \) is number of pipeline stages, \( P_b(I_s) \) is the base power consumption of an instruction at pipeline stage \( s \) and is usually obtained by putting each target instruction between two NOPs (NOP, target instruction, NOP). The term \( P_{I_{s-1}, I_s} \) is a complex term as it is affected by other instructions in the pipeline. These effects are proportional to \( h_d(I_s, I_{s-1})) \), HD of two consecutive instructions and \( h_w(I_s) \). HW of different parameters such as instruction fetch address, register number, opcode encoding, memory address, register value, and immediate operand inside each instruction. This relationship can be described by Eq. (2).

\[ P_x(I_s) \propto \beta \cdot (h_d(I_s, I_{s-1}) | h_w(I_s)) \]

In Eq. (2), \( P_x(I_s) \) is a part of power consumption for instruction \( I_s \) that is caused due to the variation in parameter \( x \) in different pipeline stages, and \( \beta \) is the variation coefficient that depends on the variation of HD or HW. Parameters with higher \( \beta \) value have larger impacts on inter-instruction power consumption compared to other parameters with the same HD/HW changes.

Depending to instruction type, each instruction may include some power-sensitive factors, and the effect of all factors involved must be considered. Therefore, \( P_i \) the power consumed during the execution of the instruction \( i \), can be calculated as:

\[ P_i = P_b(I_s) + \sum_j \beta_{i,j}N_{i,j} \]
where \( P_b(I_s) \) is the base power consumption of the instruction \( i \), \( \beta_{i,j} \) and \( N_{i,j} \) are the coefficient and the HW/HD of the \( j^{th} \) power-sensitive factor of the instruction \( i \), respectively. The processors not only dissipate power when executing instructions but also when stalling occurs. The stalling happens due to dependencies between multi cycle instructions that need more than two exe-stage cycles or stall cycles due to resource conflicts, data or control hazards. Therefore, the power consumed for instructions running in a real program can be written as Eq. (4).

\[
P_i = P_b(I_s) + \sum_j \beta_{i,j} N_{i,j} + \sum_{\in} \in
\]

where \( \in \) is the power consumption of a pipeline stall \[21\]–\[23\]. In order to find the effect of power-sensitive parameters on inter-instruction power consumption during the profiling phase, we need to create a table that shows the power consumption for each parameter at each stage of the pipeline corresponding to minimum, average and maximum HD and HW. Also, we need to sample power traces during the execution of each instruction in such a way that one parameter can change while all other parameters are kept constant. For the case of inter-instruction effects where the Hamming Distance (HD) between two consecutive instructions is a basic factor, we create a table that shows power consumption for each instruction at each pipeline stage. Then, we make a triple set of such tables, corresponding to minimum, average and maximum HD as illustrated in Fig. 2.

![FIGURE 2. Schematics of the separation of power consumption of instructions in pipelined architectures based on Hamming Distance (HD).](image)

For example, in the case of the instructions with 8-bit operands, the proper ranges of power consumptions can be obtained by changing the values of data operands from minimum to maximum values (e.g., 0 \times 00 to 0xFF) which in turn will change the Hamming distance between two consecutive operands. Inter-instruction power consumption for varying HD values of different parameters in different pipeline stages is determined with ignoring parameters with the least influence and contribution in the profiling phase, which in turn reduces the computational cost in the matching phase.

**VI. INSTRUCTION GROUPING AND CLASSIFICATION USING STATIC AND DYNAMIC POWER ANALYSIS**

The instruction set of a typical microcontroller is often categorized based on the operation of each instruction (arithmetic, logic, data, and control), its operands or register type. Since HW model is an effective model in side-channel analysis, instructions of a microcontroller can be categorized based on their HWs. Each PIC16F690 microcontroller instruction is a 14-bit word, including a 6-bit opcode and an 8-bit operand which specifies the instruction type and its operation as shown in Fig. 3 \[20\]. The grouping based on the Hamming weight of the instructions opcodes for this processor is presented in Table 2.

![FIGURE 3. General format of PIC16F690 instructions encoding.](image)

**TABLE 2. Instructions Grouping for PIC16F690 Microcontroller Based on the Operands and HW of the Opcodes.**

| Group | I | II | III |
|-------|---|----|-----|
| Type  | Arithmetic, Data | Arithmetic, data | Data |
| Operand | File register | Working register | Bit in file |
| HW of Opcode | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | NOF | MOV, CLRW | SUB, MOVF | ADWF, CLR, COMPF | INC, SUB, XOR, SWAPF | INC, DREFZ | SUB, XOR, ADD, MOV, XORLW, | BCF, BCF, BTF, BIF |
| 1 | MOV, CLRW | SUB, MOVF | ADWF, CLR, COMPF | INC, SUB, XOR, SWAPF | INC, DREFZ | SUB, XOR, ADD, MOV, XORLW, | BCF, BCF, BTF, BIF |
| 2 | MOV, CLRW | SUB, MOVF | ADWF, CLR, COMPF | INC, SUB, XOR, SWAPF | INC, DREFZ | SUB, XOR, ADD, MOV, XORLW, | BCF, BCF, BTF, BIF |
| 3 | MOV, CLRW | SUB, MOVF | ADWF, CLR, COMPF | INC, SUB, XOR, SWAPF | INC, DREFZ | SUB, XOR, ADD, MOV, XORLW, | BCF, BCF, BTF, BIF |

In this work, we used the combination of static and dynamic power analyses. Many drawbacks, which has been reported in the previous works \[21\], could be resolved by this combination. Notably, HW of a byte or word is defined as the number of ‘1’ symbols in each byte or word. Static power analysis is related to Hamming weight or the number of ‘1’s that exist on the bus and is used to recover the total HW of the target instruction, i.e., HW of opcode + HW of operand. HD between two bytes or words is the number of places at which the two bytes or words differ, and is dynamic in nature. Hence, Dynamic power analysis is employed to reveal HW of opcodes while static power analysis is applied to recover the total Hamming weight of operand and opcode. Therefore, the combination of these methods could be used for recovering the instruction types as well as the Hamming weight of their operands.

In order to analyze dynamic power of each instruction, we need to accurately record the shape of sudden and rapid changes in power consumption patterns, while other parts of power consumption are used for static power analysis. Static power analysis requires fewer samples and lower dimension,
imposes less computational burden, and can be performed at a higher speed compared with dynamic power analysis. In order to reduce the effect of noise in the experiments, measurements should be averaged over ten times. Looking at power trace of ‘ADDWF’ instructions in PIC16F690 (Fig. 4), a sharp change occurs during the rising edge of the clock where dynamic power is the dominant component. Due to the effect of capacitance discharging, this effect is diminished after falling edge of the clock and after that static power will be the dominant component. Fig. 5 shows a part of power consumption of PIC16F690 while the AES algorithm is running on the processor.

As seen in this figure, the combination of static and dynamic power analyses enables us to recover the instructions as well as their operands. Fig. 6 shows power consumption patterns including static and dynamic power consumption of ‘AND’, ‘TST’, ‘ADC’ and ‘SUB’ instructions of Cortex-M3 in one plot.

As mentioned, each PIC16F690 instruction includes an 8-bit operand, and so there are 256 possible combinations of zeros and ones. Bytes with similar HWs are categorized in 9 different HW classes: 1, 8, 28, 56, 70, 56, 28, 8, and 1 in which HW0 and HW8 with only 1 member are the smallest class and HW4 with 70 members is the largest class. These instructions with known HWs are classified by the ensemble of classifiers in their corresponding HW classes using dynamic power consumption. Ensemble learning can significantly improve the performance of pattern classification [24], [25].

In order to classify the Cortex-M3 16-bit Thumb1 instructions based on their HWs, we need to consider 17 HW classes and 33 HW classes for 32-bit Thumb2 instructions.

The general encoding formats for 16-bit and 32-bit instructions of ARM Cortex-M3 are depicted in Fig. 7 [18]. As mentioned, execution of instructions in Cortex-M3 takes 2 to 12 clock pulses. Some of the most commonly used instructions such as ‘ADD’, ‘AND’, ‘EOR’, ‘MOV’ and ‘MUL’ with 32-bit operands are performed during 1 clock cycle; some other instructions such as ‘Load’, and ‘Store’ require 2 clock cycles per instruction. ‘MUL’ with 64-bit result requires 3-5 clock cycles while ‘DIV’ timing depends on its dividend and divisor, and its execution will take 2-12 clock cycles. Table 3 categorizes the instructions of Cortex-M3 based on their required cycles as well as their corresponding bit length. Therefore, we need to distinguish cycles per instruction in ARM Cortex-M3.

The multi-cycle instructions in Cortex-M3 create pipeline stall and lead to different power consumptions, which is useful for our purpose. As we know, pipelined processors frequently insert NOP instructions to the pipeline to remove hazards, avoid data interference, and generate some delays for the proper execution of the instructions [21]–[23]. In such
In order to measure these effects on the power consumption in such circumstances, small program loops to activate the corresponding conditions could be implemented and executed. This subject is shown in Table 4 for ‘STR’ instruction as an example in which stall stages happen at cycle $n + 3$.

To distinguish the instructions of the Cortex-M3, we need to build power templates for pipeline stalls during the instruction profiling phase. Then, power consumption of the victim device in every cycle in the matching phase is compared with created templates. The pipeline stalls for the target instruction in both profiling and the matching phase in the victim device are the same, and so the stall states can be identified and determined by classification algorithms. Therefore, stall cycles for the instruction are revealed and accordingly, cycles per instruction are determined. Once the number of clock cycles required for each instruction in the victim device is determined, it is compared with the pre-prepared templates for finding the group to which that instruction belongs.

**VII. MACHINE LEARNING BASED INSTRUCTION RECOGNITION**

The aim of ML is to allow machines to learn from data so that they can produce more accurate outputs. In ML-based instruction disassembling, feature selection, dimension reduction, instruction clustering and classification are implemented through ML algorithms. Fig. 8 provides a flowchart of the entire process including the profiling and matching phases.

Feature selection is one of the fundamental concepts in machine learning which significantly impacts the performance of the selected model. The term ‘model’ describes the output of the algorithm that is trained with data. Feature selection is the process where one can manually or automatically select those features which have the highest contribution to the prediction variable or output of interest. Having irrelevant features in the data of interest can decrease the accuracy of the models and make the selected model learn based on irrelevant or inconclusive features [25], [26].

In instruction disassembling, feature selection means how useful and important each sample point in power traces is before the ML algorithm feeds on it. It is performed as a pre-processing step for faster training, reducing complexity, and improving accuracy. In this work, KL Divergence-based feature selection algorithms were employed to highlight differences between power traces of different instructions [27]. Continuous wavelet transform (CWT) was utilized to map data from time domain to frequency domain, to align power traces, and to remove noise created during the sampling [28]. Wavelet coefficients indices were used for calculating same class (instruction) KL divergence ($\text{KLD}_{SC}$) and different class (instruction) KL divergence ($\text{KLD}_{dC}$). Points with highest $\text{KLD}_{dC}$ and $\text{KLD}_{SC}$ lower than a specific threshold are selected as stationary feature points, as illustrated in algorithm I [29]. Peaks of the KL divergence exhibit distinct differences between two traces at some specific points. In real nonstationary environments, several runs of a specific program with the same instructions may lead to different values at the same sampling points/times. Therefore, such points could not be selected as feature points. Selected points were applied to PCA for more reduction as PCA eliminates low important indices without losing information data. Implemented feature selection and stationary points for ‘ADDWF’ and ‘XORWF’ instructions are shown in Fig. 9 and Fig. 10, respectively.

Using the same procedure for Cortex-M3 reduced the number of sampling points as demonstrated in Fig. 11 for ‘EOR’ instruction. In order to perform instruction clustering, the Kullback-Leibler divergences were used as a statistical metric to distinguish the distribution of static power traces of the victim device and the templates.
Algorithm 1 Stationary Features Selection (SFA) Algorithm
1: Procedure stationary features selection
2: Input: instruction power traces, \( \varphi = \{\text{Sample}_1 \ldots \text{Sample}_j\} \{\text{Class}_1 \ldots \text{Class}_i\} \)
3: Output: Set Features
4: for all class \( \epsilon \{\text{Class}_1 \ldots \text{Class}_i\} \) do
5: for all sample \( \epsilon \{\text{Sample}_1 \ldots \text{Sample}_j\} \) do
6: \( \text{WCI}_{\text{(C,} \text{S)}} \leftarrow \{\text{CWT}\{\text{C,} \text{S}}_1\ldots \text{CWT}\{\text{C,} \text{S}}_j\} \)
7: \( \text{KLD}_{\text{SC}} \leftarrow \|\text{WCI}_{\text{(C,} \text{S)}}\| \) \( \text{WCI}_{\text{(C,} \text{S)}} \)
8: \( \text{KLD}_{\text{DC}} \leftarrow \|\text{WCI}_{\text{(C,} \text{S)}}\| \) \( \text{WCI}_{\text{(C,} \text{S)}} \)
9: end for
10: end for
11: if \( \text{KLD}_{\text{SC}} < \text{KLD}_{\text{threshold}} \) and \( \text{KLD}_{\text{DC}} = \text{localmaxima} \) then
12: Set Features \( \text{Sample}_i, \text{Class}_j \)
13: end if
14: end procedure

Accuracy was calculated using Eq. (5).

\[
\text{Accuracy}\% = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \quad (5)
\]

T-SNE Machine learning algorithm uses KL divergence for visualization and clustering. This unsupervised nonlinear method maps high-dimensional data to a low-dimensional space (e.g. two-dimensional) and is able to provide well-separated clusters for instruction grouping. It minimizes the Kullback-Leibler divergence between Gaussian distribution of high dimensional space and corresponding points of Student t-distribution to a low dimensional space [30]. In this work, ensemble classification was performed for each classifier, and the results were calculated based on confusion matrix as shown in Table 5 which shows the real and predicated values in four different cases. In table 5, TP and TN represent the number of values that were correctly predicted, and FP and FN represent the number of values that were incorrectly predicated in the positive and negative classes.

| Prediction | Positive prediction | Negative prediction |
|------------|---------------------|---------------------|
| Real positive class | True Positive (TP) | False Negative (FN) |
| Real negative class | False Positive (FP) | True Negative (TN) |

VIII. ATTACK ON REAL SYSTEMS
To investigate the efficiency of the proposed approach, two dedicated boards were designed for power analysis. A 10 \( \Omega \) resistor was used to connect the ground pin of each microcontroller to the ground of the module. The power traces were recorded by Infiniium Keysight DSO90604A 20 GS/sec digital oscilloscope with BW = 6 GHz, and an Agilent SN MY 6596/4PEA, and a current probe to measure the boards.
supply currents. In order to evaluate the performance of the proposed approach, the AES algorithm was implemented in both microcontrollers. Fig. 12 shows the experimental setup for mounting the attack on PIC16F690. Fig. 13 shows the photograph of the designed LPC1768 module.

In order to improve signal-to-noise ratio and to remove trace misalignment in the time domain (or time-shift), the samples were mapped to time-scale domain using CWT transformation. The clock signal of the microcontroller was set to 8 MHz. Sampling with 1 GS/sec from a power signal that changes with 8 MHz clock frequency generates 500 sampling points per instruction which were expanded to 25,000 points per instruction in the transformed domain with scale = 50. After applying Kullback-Leibler features selection in 16 instructions of group I with optimum threshold $0.5 \times (10)^{-7}$, 1126 stationary points were selected which showed 95.4% reduction in the sampling points. The tests were performed using 5-fold cross validation: For each instruction, 500 traces were measured and were divided into five parts, where 80% of the data were used for training and 20% were used as test data. The results showed that the classification for each instruction with ensemble of classifiers was 99% successful on average.

In order to find the effect of the number of measurement traces on instruction recognition rate, the recognition rate was calculated with the total number of traces in each class. Then, the calculations were performed with fewer traces or samples. Fig. 14 shows the successful recognition rate versus the number of traces for ‘ADDWF’ instruction for both hierarchical and non-hierarchical methods. It is obvious that decreasing the number of traces reduces the recognition rate. However, the results showed that even reducing the number of traces to half leads to only 2% recognition rate reduction while reducing the number of traces to half in non-hierarchical approach leads to 11% reduction in the recognition rate.

Table 6 shows parts of real codes running on PIC16F690 as well as their 14-bit opcodes and their corresponding HWs. In order to attack the real codes, static power analysis was applied and the instructions with known HWs were classified using ensemble of classifiers in their corresponding HW classes through dynamic power consumption. Table 7 displays some of the HW4 instructions for classification of HW4 class. The Ensemble machine learning algorithm with Random forest, J48, Rep tree, Jrip and One r classifier were utilized for classification of instructions in their corresponding HW classes. Ensemble classification was performed for each classifier by WEKA tool [31] and the results were calculated based on the confusion matrix. The matrix corresponding to HW4 instructions is presented in Table 8. Fig. 15 depicts 2-D graph of the averaged power consumption of some instructions in HW4 class in which the trace for the ‘INCF’ instruction is plotted in blue, ‘RLF’ is plotted in red. As expected, all traces have almost an identical shape since they belong to the instructions that are all in the same HW class and have the same operand. In order to achieve the best possible results, we adopted the most widely known methods of ensemble including voting, bagging and adaptive boosting. Fig. 16 plots the recognition rate for PIC16F690.
TABLE 6. Parts of Real Codes Running on PIC16F690.

| Instruction  | 14 bits Opcode | HW  |
|--------------|----------------|-----|
| SUBLW 0x55   | 111100 01010101 | 8   |
| IORLW 0x55   | 110100 01010101 | 7   |
| CLRWDTC     | 000000 01010101 | 3   |
| RRF 0x40 w   | 001100 01000000 | 2   |
| BTSS PORTB1  | 011000 00000000 | 10  |
| SUBWF 0x40 w | 000010 01000000 | 2   |
| RLF 0x40 w   | 001101 01000000 | 4   |
| MOVWF 0xFF   | 110000 11111111 | 10  |
| CLR W        | 000000 00000000 | 1   |
| CLRF PORTC   | 000000 10000011 | 5   |
| ADDLW 0xAA   | 111110 11010101 | 9   |
| ADDWF 0x40 w | 000010 01000000 | 4   |

TABLE 7. Part of HW4 Class Which Were Used for Classification.

| Instruction  | 14 bits Opcode | HW  |
|--------------|----------------|-----|
| DECFSZ 0x40 w| 001011 01000000 | 4   |
| RLF 0x40 w   | 001101 01000000 | 4   |
| SWAPF 0x40 w | 001110 01000000 | 4   |
| ADDWF 0x40 w | 000111 01000000 | 4   |
| XORLW 00   | 111010 00000000 | 4   |
| INCX 0x40 f | 000011 11000000 | 4   |
| CLRF PORTB  | 000000 10000011 | 4   |
| XORWF 0 f   | 000110 11000000 | 4   |

TABLE 8. The Confusion Matrix for Some PIC16F690 Instructions in HW4 Class.

| Predicted instructions | Random Forest | J48 | Rep tree | Jrip | One r |
|------------------------|---------------|-----|---------|-----|-------|
| RLF 0x40, w            | 8             | 453 | 10      | 12  | 6     |
| ADDWF x40, w           | 5             | 6   | 6       | 459 | 4     |

‘ADDWF’ and ‘RLF’ instructions. It clearly shows that the recognition rate is considerably improved via hierarchical ensemble classification along with the adaptive boosting, bagging and weighted voting. As evident in Fig. 16, without hierarchical and ensemble classification, the highest recognition rate for ‘ADDWF/RLF’ instruction in HW-4 class are 47% and 44.6%, respectively, which belong to Random forest and Rep tree algorithms. By applying hierarchical approach, the highest recognition rate improves 25.4% and 24.2% using J48 and Random forest algorithms, respectively. By applying bagging, the recognition rate rises while the highest improvement rate is 14.2% and 13.6%, which belongs to Boosting Jrip and One r algorithms, respectively. By the application of boosting, recognition rate increases and the highest rate of improvement is 20.6% and 18.8% owned by boosting random forest and J48 algorithms. Finally, by applying the weighted majority voting, it reaches 91.8% and 90.6% for the same instructions. This process was carried out for other instructions in the other HW classes. Table 9 presents instruction recognition rate for some other instructions with other HWs that change from 88.4% to 98%. The results show that a considerable improvement has been obtained in recognition of the instructions in a real program with 93.3% success rate on average.

It should also be mentioned that, as KL Divergence-based feature selection algorithms are employed to select non-stationary feature points, we should find the optimum value for KL threshold to achieve the best recognition rate, the minimum selected points, the maximum feature reduction, and the minimum computational cost. In Fig. 17, the recognition rate and trace reduction versus the KL threshold value are illustrated. As seen in Fig. 17, the intersection of the recognition rate and trace reduction curves gives the optimum KL value which is equal to 0.5.

Applying the same procedure as was used for PIC, 946 stationary points were obtained for Cortex-M3 which showed 96.2% reduction in the number of sampling points. The stationary points were applied as input variables to the minimum selected points, the maximum feature reduction, and the minimum computational cost. In Fig. 17, the recognition rate and trace reduction versus the KL threshold value are illustrated. As seen in Fig. 17, the intersection of the recognition rate and trace reduction curves gives the optimum KL value which is equal to 0.5.

Applying the same procedure as was used for PIC, 946 stationary points were obtained for Cortex-M3 which showed 96.2% reduction in the number of sampling points. The stationary points were applied as input variables to the minimum selected points, the maximum feature reduction, and the minimum computational cost. In Fig. 17, the recognition rate and trace reduction versus the KL threshold value are illustrated. As seen in Fig. 17, the intersection of the recognition rate and trace reduction curves gives the optimum KL value which is equal to 0.5.

Applying the same procedure as was used for PIC, 946 stationary points were obtained for Cortex-M3 which showed 96.2% reduction in the number of sampling points. The stationary points were applied as input variables to the minimum selected points, the maximum feature reduction, and the minimum computational cost. In Fig. 17, the recognition rate and trace reduction versus the KL threshold value are illustrated. As seen in Fig. 17, the intersection of the recognition rate and trace reduction curves gives the optimum KL value which is equal to 0.5.

Applying the same procedure as was used for PIC, 946 stationary points were obtained for Cortex-M3 which showed 96.2% reduction in the number of sampling points. The stationary points were applied as input variables to the minimum selected points, the maximum feature reduction, and the minimum computational cost. In Fig. 17, the recognition rate and trace reduction versus the KL threshold value are illustrated. As seen in Fig. 17, the intersection of the recognition rate and trace reduction curves gives the optimum KL value which is equal to 0.5.
PCA algorithm for more reduction and resulted in 98.2% recognition rate for test codes using 50 components in 5-fold cross-validation. The result of attack on some of the selected 16-bit thumb instructions of LPC1768 ARM Cortex-M3 with immediate offsets and register operands in the real program are shown in Table 10. The results showed 25.5% improvement in recognition rate on average after applying the proposed hierarchical approach. Fig. 18 shows the results of successful recognition rate versus the number of traces for both hierarchical and non-hierarchical methods. Similar to the PIC, decreasing the number of traces reduces the recognition rate, but according to the results even reducing the number of traces to half in hierarchical method leads to only 4% recognition rate reduction. However, reducing the number of traces to half in non-hierarchical approach leads to 15% reduction in the recognition rate.

### IX. SCALABILITY AND COMPUTATIONAL COST

Scalability is an essential component of any state-of-the-art design and implementation. It leads to better user experience, lower maintenance costs, and higher agility. Our design is scalable in terms of both hardware and software. With respect to the general scenario of the proposed approach, it consists of four main components: a target processor (in this work, PIC16F690 as an 8-bit and ARM Cortex M-3 as a 32-bit platform), hierarchical grouping of the instructions based on their Hamming weight, building power templates for each instruction power analysis, and template matching using machine learning approaches.

The software architecture of our customized simulation and attack environment can accommodate various scenarios depending on the processor type or model. It should be noted that other than the two processors studied in this work, the proposed methodology would be applicable for both 8-bit cores and (16 and) 32-bit cores of the same families or even different families. It would also be applicable for various security algorithms, specifically symmetric key cryptographic algorithms. The sampling process and template building phase are performed offline. Hence, changing the target processor will not impose any computational burden on the work. In this case, we may only need to change the sampling frequency of the digital oscilloscope. After the sampling process, the input data and their associated power traces are paired, stored and exported to the analysis and preprocessing program written in MATLAB, and also to WEKA tools for machine learning-based analysis. This process is also offline and quite flexible in such a way that it can be changed and updated based on the attack scenario.

On the other hand, our proposed attack framework is much more efficient in terms of computational cost compared with the other methods published in the literature due to the following reasons:

a) The most important factor in the amount of calculations is the number of required power samples as it directly affects time consumption and storage volume required for the profiling phase. The large profiling base required to successfully construct an accurate template is still an open problem in the existing profiling attacks. This problem is deteriorated by the necessity for the target and profiling
TABLE 11. Runtime Comparison for Different Classification Algorithms.

| Method          | Ave. Training Time (Sec.) | Average Classification Time (Sec.) |
|-----------------|---------------------------|-----------------------------------|
| Random Forest   | 5.12                      | 0.091                             |
| 3AS             | 1.17                      | 0.083                             |
| REPTree         | 1.89                      | 0.086                             |
| JRP             | 9.96                      | 0.079                             |
| Oner            | 1.42                      | 0.081                             |
| Ensemble of Classifiers | 11.39          | 0.090                             |

TABLE 12. Comparison Between the Proposed Method and Other Similar Works.

| Ref. | Analysis Method | Target Microcontroller | Classifier | Test Codes Identification | Real codes Identification |
|------|-----------------|------------------------|------------|---------------------------|----------------------------|
| [4]  | Dynamic Power   | ATMEL AVR Xmega        | Statistical| 85%                       | Not Evaluated              |
| [8]  | Dynamic Power   | PIC16F687               | Multivariate Gaussian | 70% | 58% |
| [9]  | Dynamic Power   | ATMEGA 163             | KNN        | 100%                      | Not Evaluated              |
| [10] | Multiple EM     | PIC16F687               | KNN        | 96.24%                    | 87.69%                     |
| [11] | EM              | ATMEGA 163             | KNN, PCA   | 78.3%                     | Not Evaluated              |
| [12, 13]| Dynamic Power   | ATMEGA 224F           | QDA, SVM   | 99.03%                    | Not Evaluated              |
| [14] | Multiple EM     | PIC16F153               | QDA        | 95% for 14 bit            | Not Evaluated              |
| [17] | Dynamic Power   | ARM Cortex-M0          | Linear Regression | -  | -  |
| Our   | Static and Dynamic Power Analysis | PIC16F690 | Ensemble of Classifiers | 99.5% | 93.3% |
| Work | ARM Cortex-M3   |                        |            | 98%                       | 80.2%                      |

In this work, we demonstrated how to use hierarchical and ensemble classification along with dynamic and static power analyses to increase the average rate of extraction for test codes running on PIC16F690 and Hamming weight of their corresponding operands up to 99.5%. We also performed the same procedure for ARM-based LPC1768 and achieved 98% extraction rate. In real experiments on real codes running on these devices, 93% success rate was obtained on average for PIC16F690 and 80% for LPC1768. This means that the proposed method works for both a mid-range core and a relatively high-end core, and is portable between different chips of the same family or even different families. In the field of hardware and cyber security, any improvement in the accuracy of machine learning algorithms can be of great value. ML-based template attacks are less prone to some of the weaknesses found in statistical-based attacks and outperform them in scenarios where complicating factors such as ambient factors or countermeasure mechanisms are present, specifically when a sufficient profiling base is available. These methods are also successful for smaller profiling traces. The result of this work can be applied for malware detection where an adversary can insert a malicious code with similar functionality as the original code in the device. It should also be noted that, as the dimensions of CMOS transistors decreases to few nanometers, the role of static power consumption is becoming increasingly important in modern devices and low-power applications. This is a central issue from the security point of view since it is being converted to a new target for hardware and side-channel attackers. An advanced organized security attack could take over the control of power grids, energy providers, financial services, or other critical infrastructures, resulting in catastrophic consequences. Enhancing hardware and cyber security is perhaps the fundamental component to protect such infrastructures. As a suggestion for future work, other processors in more advanced technology generations need to be scrutinized. In addition, deep-learning-based approaches may provide better results over commonly used ML-techniques [32].

is usually performed offline. So, ensemble of classifiers is independent of matching phase. Table 11 provides a comparison between the execution time of different classification algorithms on the same platform.

X. COMPARISON WITH OTHER WORKS

Table 12 presents a comparison between some of the existing side-channel based disassemblers with our work. As mentioned, most of the existing works have not evaluated their approaches for real codes. Some of them use EM analysis which requires decapsulating facilities with several extra processes, and also needs a motorized probe as well as precise experimental setup that are available only in special laboratories. There are a few references about reverse engineering of the ARM processor using power analysis attacks none of which present an actual disassembler to recover real instructions from non-invasive measurements.

XI. CONCLUSION AND FUTURE WORKS

In this work, we demonstrated how to use hierarchical and ensemble classification along with dynamic and static power analyses to increase the average rate of extraction for test codes running on PIC16F690 and Hamming weight of their corresponding operands up to 99.5%. We also performed the same procedure for ARM-based LPC1768 and achieved 98% extraction rate. In real experiments on real codes running on these devices, 93% success rate was obtained on average for PIC16F690 and 80% for LPC1768. This means that the proposed method works for both a mid-range core and a relatively high-end core, and is portable between different chips of the same family or even different families. In the field of hardware and cyber security, any improvement in the accuracy of machine learning algorithms can be of great value. ML-based template attacks are less prone to some of the weaknesses found in statistical-based attacks and outperform them in scenarios where complicating factors such as ambient factors or countermeasure mechanisms are present, specifically when a sufficient profiling base is available. These methods are also successful for smaller profiling traces. The result of this work can be applied for malware detection where an adversary can insert a malicious code with similar functionality as the original code in the device. It should also be noted that, as the dimensions of CMOS transistors decreases to few nanometers, the role of static power consumption is becoming increasingly important in modern devices and low-power applications. This is a central issue from the security point of view since it is being converted to a new target for hardware and side-channel attackers. An advanced organized security attack could take over the control of power grids, energy providers, financial services, or other critical infrastructures, resulting in catastrophic consequences. Enhancing hardware and cyber security is perhaps the fundamental component to protect such infrastructures. As a suggestion for future work, other processors in more advanced technology generations need to be scrutinized. In addition, deep-learning-based approaches may provide better results over commonly used ML-techniques [32].
REFERENCES

[1] M. Jahanbani, Z. Noroozi, and N. Bagheri, “DPA protected implementation of OCB and COLM authenticated ciphers,” IEEE Access, vol. 7, pp. 139815–139826, 2019.

[2] M. Masoumi, “A highly efficient and secure hardware implementation of the advanced encryption standard,” J. Inf. Secur. Appl., vol. 48, Oct. 2019, pp. 102371.

[3] S. C. Seo and H. Kim, “SCA-resistant GCM implementation on B-IT AVR microcontrollers,” IEEE Access, vol. 7, pp. 103961–103978, 2019.

[4] M. O. Choudary and M. G. Kuhn, “Efficient, portable template attacks,” IEEE Trans. Inf. Forensics Security, vol. 13, no. 2, pp. 490–501, Feb. 2018.

[5] Y.-S. Won, B.-Y. Sim, and J.-Y. Park, “Key schedule against template attack-based simple power analysis on a single target,” Appl. Sci., vol. 10, no. 11, p. 3804, May 2020, doi: 10.3390/app10113804.

[6] S. Picek, A. Heuser, A. Jovic, K. Knezevic, and T. Richmond, “Improving side-channel analysis through semi-supervised learning,” in Smart Card Research and Advanced Applications. CARDIS (Lecture Notes in Computer Science), vol. 11389, B. Bilgin and J. B. Fischer, Eds. Cham, Switzerland: Springer, 2019, pp. 78–99.

[7] M. Goldak, “Side-Channel based reverse engineering for microcontrolers,” in Computer Architecture: A Quantitative Approach, 6th ed. Burlington, MD, USA: Morgan-Kaufmann, 2017.

[8] P. Lotfi-Kamran, A.-M. Rahmani, A.-A. Salehpour, A. Afazali-kousha, and Z. Navabi, “Dynamic Power Reduction of Stalls in Pipelined Architecture Processors,” Int. J. Design. Anal. Tools Circuits Syst., vol. 1, no. 1, pp. 9–15, Jun. 2011.

[9] S. Picek, A. Heuser, A. Jovic, K. Knezevic, and T. Richmond, “Improving side-channel analysis through semi-supervised learning,” in Smart Card Research and Advanced Applications. CARDIS (Lecture Notes in Computer Science), vol. 11389, B. Bilgin and J. B. Fischer, Eds. Cham, Switzerland: Springer, 2019, pp. 78–99.

[10] M. Masoumi, “A highly efficient and secure hardware implementation of the advanced encryption standard,” J. Inf. Secur. Appl., vol. 48, Oct. 2019, pp. 102371.